

# СПРАВОЧНИК по языку программирования MQL5

## для клиентского терминала MetaTrader 5

ИЗУЧИТЕ язык MQL5  
и РЕШАЙТЕ любые задачи:

- Создание собственных индикаторов технического анализа любой сложности
- Автотрейдинг - автоматизация торговой системы для работы на разнообразных финансовых рынках
- Разработка аналитических инструментов на основе математических достижений и классических методов
- Написание информационно-торговых систем для широкого круга задач (трейдинг, мониторинг, сигналы и т.д.)

# Содержание

## Справочник MQL5

67

1 Основы языка .....	69
Синтаксис .....	70
Комментарии.....	71
Идентификаторы.....	72
Зарезервированные слова.....	73
Типы данных .....	74
Целые типы.....	75
Типы char, short, int и long .....	76
Символьные константы .....	80
Тип datetime.....	84
Тип color .....	85
Тип bool .....	86
Перечисления.....	87
Вещественные типы (double, float).....	89
Тип string.....	94
Структуры, классы и интерфейсы.....	95
Объект динамического массива.....	120
Приведение типов.....	122
Тип void и константа NULL.....	127
Пользовательские типы.....	128
Указатели объектов.....	138
Ссылки. Модификатор & и ключевое слово this .....	140
Операции и выражения .....	142
Выражения.....	143
Арифметические операции.....	144
Операции присваивания.....	145
Операции отношения.....	146
Логические операции.....	147
Побитовые операции.....	149
Другие операции.....	152
Приоритеты и порядок операций.....	156
Операторы .....	158
Составной оператор .....	160
Оператор-выражение.....	161
Оператор возврата return.....	162
Условный оператор if-else.....	163
Условный оператор ?: .....	164
Оператор-переключатель switch.....	166
Оператор цикла while .....	168
Оператор цикла for.....	169
Оператор цикла do while .....	171
Оператор завершения break .....	172
Оператор продолжения continue .....	173
Оператор создания объекта new .....	174
Оператор уничтожения объекта delete .....	176
Функции .....	177
Вызов функции.....	179
Передача параметров.....	180
Перегрузка функций.....	183
Перегрузка операций .....	186
Описание внешних функций.....	200
Экспортирование функций.....	202

Функции обработки событий.....	203
<b>Переменные .....</b>	<b>215</b>
Локальные переменные.....	218
Формальные параметры .....	220
Статические переменные.....	222
Глобальные переменные.....	224
Input переменные.....	225
Extern переменные.....	230
Инициализация переменных .....	231
Область видимости и время жизни переменных.....	233
Создание и уничтожение объектов.....	235
<b>Препроцессор .....</b>	<b>238</b>
Макроподстановка (#define).....	240
Свойства программ (#property).....	243
Включение файлов (#include).....	250
Импорт функций (#import).....	251
Условная компиляция (#ifdef, #ifndef, #else, #endif).....	254
<b>Объектно-ориентированное программирование .....</b>	<b>256</b>
Инкапсуляция и расширяемость типов .....	258
Наследование.....	261
Полиморфизм.....	266
Перегрузка.....	270
Виртуальные функции .....	271
Статические члены класса.....	275
Шаблоны функций.....	279
Шаблоны классов.....	284
Абстрактные классы .....	289
<b>2 Константы, перечисления и структуры.....</b>	<b>291</b>
<b>Константы графиков .....</b>	<b>292</b>
Типы событий графика.....	293
Периоды графиков.....	300
Свойства графиков.....	302
Позиционирование графика.....	311
Отображение графиков .....	312
Примеры работы с графиком .....	314
<b>Константы объектов .....</b>	<b>373</b>
Типы объектов .....	374
OBJ_VLINE .....	376
OBJ_HLINE .....	381
OBJ_TREND .....	386
OBJ_TRENDBYANGLE .....	393
OBJ_CYCLES.....	399
OBJ_ARROWED_LINE.....	405
OBJ_CHANNEL.....	411
OBJ_STDDEVCHANNEL.....	418
OBJ_REGRESSION.....	425
OBJ_PITCHFORK.....	431
OBJ_GANNLINE.....	439
OBJ_GANNFAN.....	446
OBJ_GANNGRID.....	453
OBJ_FIBO .....	460
OBJ_FIBOTIMES .....	467
OBJ_FIBOFAN.....	474
OBJ_FIBOARC.....	481
OBJ_FIBOCHANNEL.....	488
OBJ_EXPANSION.....	496
OBJ_ELLIOTWAVE5.....	504
OBJ_ELLIOTWAVE3.....	512

OBJ_RECTANGLE .....	519
OBJ_TRIANGLE.....	525
OBJ_ELLIPSE.....	532
OBJ_ARROW_THUMB_UP.....	538
OBJ_ARROW_THUMB_DOWN.....	544
OBJ_ARROW_UP.....	550
OBJ_ARROW_DOWN.....	556
OBJ_ARROW_STOP.....	562
OBJ_ARROW_CHECK.....	568
OBJ_ARROW_LEFT_PRICE.....	574
OBJ_ARROW_RIGHT_PRICE.....	579
OBJ_ARROW_BUY.....	584
OBJ_ARROW_SELL.....	589
OBJ_ARROW.....	594
OBJ_TEXT .....	600
OBJ_LABEL .....	606
OBJ_BUTTON.....	614
OBJ_CHART .....	621
OBJ_BITMAP.....	628
OBJ_BITMAP_LABEL.....	635
OBJ_EDIT .....	642
OBJ_EVENT .....	649
OBJ_RECTANGLE_LABEL.....	654
Свойства объектов .....	660
Способы привязки объектов.....	668
Угол привязки .....	673
Видимость объектов .....	676
Уровни волн Эллиотта.....	679
Объекты Ганна.....	680
Набор Web-цветов.....	682
Wingdings.....	684
<b>Константы индикаторов .....</b>	<b>685</b>
Ценовые константы.....	686
Методы скользящих .....	689
Линии индикаторов.....	690
Стили рисования .....	692
Свойства пользовательских индикаторов .....	696
Типы индикаторов.....	699
Идентификаторы типов данных .....	701
<b>Состояние окружения .....</b>	<b>702</b>
Состояние клиентского терминала.....	703
Информация о запущенной MQL5-программе .....	709
Информация об инструменте.....	712
Информация о счете.....	730
Статистика тестирования.....	735
<b>Торговые константы .....</b>	<b>739</b>
Информация об исторических данных по инструменту.....	740
Свойства ордеров.....	741
Свойства позиций.....	746
Свойства сделок.....	749
Типы торговых операций .....	753
Типы торговых транзакций .....	763
Виды заявок в стакане цен .....	766
Свойства сигналов.....	767
<b>Именованные константы .....</b>	<b>769</b>
Предопределенные макроподстановки.....	770
Математические константы.....	772
Константы числовых типов .....	774

Причины деинициализации .....	777
Проверка указателя объекта.....	779
Прочие константы .....	780
<b>Структуры данных .....</b>	<b>784</b>
Структура даты .....	785
Структура входных параметров индикатора.....	786
Структура исторических данных.....	787
Структура стакана цен.....	788
Структура торгового запроса.....	789
Структура результата проверки торгового запроса.....	803
Структура результата торгового запроса .....	804
Структура торговой транзакции.....	808
Структура для получения текущих цен .....	816
Структуры экономического календаря.....	818
<b>Коды ошибок и предупреждений .....</b>	<b>823</b>
Коды возврата торгового сервера .....	824
Предупреждения компилятора.....	828
Ошибки компиляции.....	832
Ошибки времени выполнения.....	845
<b>Константы ввода/вывода .....</b>	<b>858</b>
Флаги открытия файлов.....	859
Свойства файлов.....	862
Позиционирование внутри файла.....	863
Использование кодовой страницы .....	864
MessageBox.....	865
<b>3 Программы MQL5.....</b>	<b>867</b>
Выполнение программ .....	868
Разрешение на торговлю .....	876
События клиентского терминала .....	880
Ресурсы .....	883
Вызов импортируемых функций .....	895
Ошибки выполнения .....	897
Тестирование торговых стратегий .....	898
<b>4 Предопределенные переменные.....</b>	<b>925</b>
__AppliedTo .....	926
__Digits .....	928
__Point .....	929
__LastError .....	930
__Period .....	931
__RandomSeed .....	932
__StopFlag .....	933
__Symbol .....	934
__UninitReason .....	935
__IsX64 .....	936
<b>5 Общие функции .....</b>	<b>937</b>
Alert .....	939
CheckPointer .....	940
Comment .....	942
CryptEncode .....	944
CryptDecode .....	946
DebugBreak .....	947
ExpertRemove .....	948
GetPointer .....	950
GetTickCount .....	954
GetMicrosecondCount .....	956
MessageBox .....	958
PeriodSeconds .....	959

PlaySound .....	960
Print .....	961
PrintFormat .....	963
ResetLastError .....	970
ResourceCreate .....	971
ResourceFree .....	973
ResourceReadImage .....	974
ResourceSave .....	975
SetReturnError .....	976
SetUserError .....	977
Sleep .....	978
TerminalClose .....	979
TesterHideIndicators .....	981
TesterStatistics .....	983
TesterStop .....	984
TesterDeposit .....	985
TesterWithdrawal .....	986
TranslateKey .....	987
ZeroMemory .....	988
<b>6 Операции с массивами.....</b>	<b>989</b>
ArrayBsearch .....	991
ArrayCopy .....	995
ArrayCompare .....	1000
ArrayFree .....	1001
ArrayGetAsSeries .....	1010
ArrayInitialize .....	1013
ArrayFill .....	1015
ArrayIsDynamic .....	1017
ArrayIsSeries .....	1019
ArrayMaximum .....	1021
ArrayMinimum .....	1032
ArrayPrint .....	1043
ArrayRange .....	1046
ArrayResize .....	1047
ArrayInsert .....	1050
ArrayRemove .....	1053
ArrayReverse .....	1055
ArraySetAsSeries .....	1057
ArraySize .....	1060
ArraySort .....	1062
ArraySwap .....	1067
<b>7 Преобразование данных.....</b>	<b>1069</b>
CharToString .....	1071
CharArrayToString .....	1072
CharArrayToStruct .....	1073
StructToCharArray .....	1074
ColorToARGB .....	1075
ColorToString .....	1077
DoubleToString .....	1078
EnumToString .....	1079
IntegerToString .....	1081
ShortToString .....	1082
ShortArrayToString .....	1083
TimeToString .....	1084
NormalizeDouble .....	1085
StringToCharArray .....	1087
StringToColor .....	1088
StringtoDouble .....	1089

StringToInteger .....	1090
StringToShortArray .....	1091
StringToTime .....	1092
StringFormat .....	1093
<b>8 Математические функции.....</b>	<b>1097</b>
MathAbs .....	1099
MathArccos .....	1100
MathArcsin .....	1101
MathArctan .....	1102
MathCeil .....	1103
MathCos .....	1104
MathExp .....	1105
MathFloor .....	1106
MathLog .....	1107
MathLog10 .....	1108
MathMax .....	1109
MathMin .....	1110
MathMod .....	1111
MathPow .....	1112
MathRand .....	1113
MathRound .....	1114
MathSin .....	1115
MathSqrt .....	1116
MathStrand .....	1117
MathTan .....	1120
MathIsValidNumber .....	1121
MathExprm1 .....	1122
MathLog1p .....	1123
MathArccosh .....	1124
MathArcsinh .....	1125
MathArctanh .....	1126
MathCosh .....	1127
MathSinh .....	1128
MathTanh .....	1129
MathSwap .....	1130
<b>9 Строковые функции.....</b>	<b>1131</b>
StringAdd .....	1133
StringBufferLen .....	1135
StringCompare .....	1136
StringConcatenate .....	1138
StringFill .....	1139
StringFind .....	1140
StringGetCharacter .....	1141
StringInit .....	1142
StringLen .....	1143
StringSetLength .....	1144
StringReplace .....	1145
StringReserve .....	1146
StringSetCharacter .....	1148
StringSplit .....	1150
StringSubstr .....	1152
StringToLower .....	1153
StringToUpper .....	1154
StringTrimLeft .....	1155
StringTrimRight .....	1156
<b>10 Дата и время.....</b>	<b>1157</b>
TimeCurrent .....	1158

TimeTradeServer .....	1159
TimeLocal .....	1160
TimeGMT .....	1161
TimeDaylightSavings .....	1162
TimeGMTOffset .....	1163
TimeToStruct .....	1164
StructToTime .....	1165
<b>11 Информация о счете.....</b>	<b>1166</b>
AccountInfoDouble .....	1167
AccountInfoInteger .....	1168
AccountInfoString .....	1170
<b>12 Проверка состояния.....</b>	<b>1171</b>
GetLastError .....	1172
IsStopped .....	1173
UninitializeReason .....	1174
TerminalInfoInteger .....	1175
TerminalInfoDouble .....	1176
TerminalInfoString .....	1177
MQLInfoInteger .....	1178
MQLInfoString .....	1179
Symbol .....	1180
Period .....	1181
Digits .....	1182
Point .....	1183
<b>13 Обработка событий.....</b>	<b>1184</b>
OnStart .....	1186
OnInit .....	1189
OnDeinit .....	1192
OnTick .....	1195
OnCalculate .....	1201
OnTimer .....	1205
OnTrade .....	1208
OnTradeTransaction .....	1213
OnBookEvent .....	1219
OnChartEvent .....	1223
OnTester .....	1230
OnTesterInit .....	1237
OnTesterDeinit .....	1244
OnTesterPass .....	1245
<b>14 Получение рыночной информации.....</b>	<b>1246</b>
SymbolsTotal .....	1248
SymbolExist .....	1249
SymbolName .....	1250
SymbolSelect .....	1251
SymbolIsSynchronized .....	1252
SymbolInfoDouble .....	1253
SymbolInfoInteger .....	1255
SymbolInfoString .....	1257
SymbolInfoMarginRate .....	1258
SymbolInfoTick .....	1259
SymbolInfoSessionQuote .....	1260
SymbolInfoSessionTrade .....	1261
MarketBookAdd .....	1262
MarketBookRelease .....	1263
MarketBookGet .....	1264
<b>15 Экономический календарь.....</b>	<b>1265</b>

CalendarCountryById .....	1266
CalendarEventById .....	1268
CalendarValueById .....	1271
CalendarCountries .....	1274
CalendarEventByCountry .....	1276
CalendarEventByCurrency .....	1278
CalendarValueHistoryByEvent .....	1280
CalendarValueHistory .....	1283
CalendarValueLastByEvent .....	1286
CalendarValueLast .....	1291
<b>16 Доступ к таймсериалам и индикаторам.....</b>	<b>1296</b>
Направление индексации в массивах и таймсериев .....	1301
Организация доступа к данным .....	1305
SeriesInfoInteger .....	1315
Bars .....	1317
BarsCalculated .....	1320
IndicatorCreate .....	1322
IndicatorParameters .....	1325
IndicatorRelease .....	1327
CopyBuffer .....	1329
CopyRates .....	1334
CopyTime .....	1338
CopyOpen .....	1341
CopyHigh .....	1344
CopyLow .....	1348
CopyClose .....	1351
CopyTickVolume .....	1354
CopyRealVolume .....	1358
CopySpread .....	1361
CopyTicks .....	1365
CopyTicksRange .....	1371
iBars .....	1373
iBarShift .....	1374
iClose .....	1377
iHigh .....	1379
iHighest .....	1381
iLow .....	1382
iLowest .....	1384
iOpen .....	1385
iTime .....	1387
iTICKVolume .....	1390
iRealVolume .....	1392
iVolume .....	1394
iSpread .....	1396
<b>17 Пользовательские символы.....</b>	<b>1398</b>
CustomSymbolCreate .....	1400
CustomSymbolDelete .....	1402
CustomSymbolSetInteger .....	1403
CustomSymbolSetDouble .....	1404
CustomSymbolSetString .....	1405
CustomSymbolSetMarginRate .....	1406
CustomSymbolSetSessionQuote .....	1407
CustomSymbolSetSessionTrade .....	1408
CustomRatesDelete .....	1409
CustomRatesReplace .....	1410
CustomRatesUpdate .....	1411
CustomTicksAdd .....	1412
CustomTicksDelete .....	1414

CustomTicksReplace .....	1415
CustomBookAdd .....	1417
<b>18 Операции с графиками.....</b>	<b>1420</b>
ChartApplyTemplate .....	1423
ChartSaveTemplate .....	1426
ChartWindowFind .....	1431
ChartTimePriceToXY .....	1433
ChartXYToTimePrice .....	1434
ChartOpen .....	1436
ChartFirst .....	1437
ChartNext .....	1438
ChartClose .....	1439
ChartSymbol .....	1440
ChartPeriod .....	1441
ChartRedraw .....	1442
ChartSetDouble .....	1443
ChartSetInteger .....	1444
ChartSetString .....	1446
ChartGetDouble .....	1448
ChartGetInteger .....	1450
ChartGetString .....	1452
ChartNavigate .....	1454
ChartID .....	1457
ChartIndicatorAdd .....	1458
ChartIndicatorDelete .....	1462
ChartIndicatorGet .....	1465
ChartIndicatorName .....	1467
ChartIndicatorsTotal .....	1468
ChartWindowOnDropped .....	1469
ChartPriceOnDropped .....	1470
ChartTimeOnDropped .....	1471
ChartXOnDropped .....	1472
ChartYOnDropped .....	1473
ChartSetSymbolPeriod .....	1474
ChartScreenShot .....	1475
<b>19 Торговые функции.....</b>	<b>1478</b>
OrderCalcMargin .....	1481
OrderCalcProfit .....	1482
OrderCheck .....	1483
OrderSend .....	1484
OrderSendAsync .....	1489
PositionsTotal .....	1500
PositionGetSymbol .....	1501
PositionSelect .....	1502
PositionSelectByTicket .....	1503
PositionGetDouble .....	1504
PositionGetInteger .....	1505
PositionGetString .....	1507
PositionGetTicket .....	1508
OrdersTotal .....	1509
OrderGetTicket .....	1510
OrderSelect .....	1512
OrderGetDouble .....	1513
OrderGetInteger .....	1514
OrderGetString .....	1515
HistorySelect .....	1516
HistorySelectByPosition .....	1518
HistoryOrderSelect .....	1519

HistoryOrdersTotal .....	1520
HistoryOrderGetTicket .....	1521
HistoryOrderGetDouble .....	1523
HistoryOrderGetInteger .....	1524
HistoryOrderGetString .....	1527
HistoryDealSelect .....	1528
HistoryDealsTotal .....	1529
HistoryDealGetTicket .....	1530
HistoryDealGetDouble .....	1532
HistoryDealGetInteger .....	1533
HistoryDealGetString .....	1536
<b>20 Управление сигналами.....</b>	<b>1537</b>
SignalBaseGetDouble .....	1538
SignalBaseGetInteger .....	1539
SignalBaseGetString .....	1540
SignalBaseSelect .....	1541
SignalBaseTotal .....	1542
SignalInfoGetDouble .....	1543
SignalInfoGetInteger .....	1544
SignalInfoGetString .....	1545
SignalInfoSetDouble .....	1546
SignalInfoSetInteger .....	1547
SignalSubscribe .....	1548
SignalUnsubscribe .....	1549
<b>21 Сетевые функции.....</b>	<b>1550</b>
SocketCreate .....	1552
SocketClose .....	1555
SocketConnect .....	1558
SocketIsConnected .....	1562
SocketIsReadable .....	1563
SocketIsWritable .....	1566
SocketTimeouts .....	1567
SocketRead .....	1568
SocketSend .....	1572
SocketTlsHandshake .....	1576
SocketTlsCertificate .....	1577
SocketTlsRead .....	1581
SocketTlsReadAvailable .....	1585
SocketTlsSend .....	1586
WebRequest .....	1587
SendFTP .....	1590
SendMail .....	1591
SendNotification .....	1592
<b>22 Глобальные переменные терминала.....</b>	<b>1593</b>
GlobalVariableCheck .....	1594
GlobalVariableTime .....	1595
GlobalVariableDel .....	1596
GlobalVariableGet .....	1597
GlobalVariableName .....	1598
GlobalVariableSet .....	1599
GlobalVariablesFlush .....	1600
GlobalVariableTemp .....	1601
GlobalVariableSetOnCondition .....	1602
GlobalVariablesDeleteAll .....	1603
GlobalVariablesTotal .....	1604
<b>23 Файловые операции.....</b>	<b>1605</b>
FileFindFirst .....	1608

FileFindNext .....	1610
FileFindClose .....	1612
FileIsExist .....	1614
FileOpen .....	1617
FileClose .....	1620
FileCopy .....	1621
FileDelete .....	1624
FileMove .....	1626
FileFlush .....	1628
FileGetInteger .....	1630
FileIsEnding .....	1633
FileIsLineEnding .....	1635
FileReadArray .....	1640
FileReadBool .....	1642
FileReadDatetime .....	1645
FileReadDouble .....	1648
FileReadFloat .....	1651
FileReadInteger .....	1654
FileReadLong .....	1658
FileReadNumber .....	1661
FileReadString .....	1666
FileReadStruct .....	1668
FileSeek .....	1672
FileSize .....	1675
FileTell .....	1677
FileWrite .....	1680
FileWriteArray .....	1683
FileWriteDouble .....	1686
FileWriteFloat .....	1689
FileWriteInteger .....	1691
FileWriteLong .....	1694
FileWriteString .....	1696
FileWriteStruct .....	1699
FileLoad .....	1702
FileSave .....	1704
FolderCreate .....	1706
FolderDelete .....	1709
FolderClean .....	1712
<b>24 Пользовательские индикаторы.....</b>	<b>1715</b>
Стили индикаторов в примерах .....	1719
DRAW_NONE .....	1727
DRAW_LINE .....	1730
DRAW_SECTION .....	1734
DRAW_HISTOGRAM .....	1738
DRAW_HISTOGRAM2 .....	1742
DRAW_ARROW .....	1746
DRAW_ZIGZAG .....	1751
DRAW_FILLING .....	1756
DRAW_BARS .....	1761
DRAW_CANDLES .....	1767
DRAW_COLOR_LINE .....	1774
DRAW_COLOR_SECTION .....	1779
DRAW_COLOR_HISTOGRAM .....	1785
DRAW_COLOR_HISTOGRAM2 .....	1790
DRAW_COLOR_ARROW .....	1795
DRAW_COLOR_ZIGZAG .....	1801
DRAW_COLOR_BARS .....	1806
DRAW_COLOR_CANDLES .....	1813

Связь между свойствами индикатора и функциями .....	1820
SetIndexBuffer .....	1823
IndicatorSetDouble .....	1826
IndicatorSetInteger .....	1830
IndicatorSetString .....	1834
PlotIndexSetDouble .....	1837
PlotIndexSetInteger .....	1838
PlotIndexSetString .....	1842
PlotIndexGetInteger .....	1843
<b>25 Графические объекты.....</b>	<b>1846</b>
ObjectCreate .....	1848
ObjectName .....	1852
ObjectDelete .....	1853
ObjectsDeleteAll .....	1854
ObjectFind .....	1855
ObjectGetTimeByValue .....	1856
ObjectGetValueByTime .....	1857
ObjectMove .....	1858
ObjectsTotal .....	1859
ObjectSetDouble .....	1860
ObjectSetInteger .....	1864
ObjectSetString .....	1867
ObjectGetDouble .....	1869
ObjectGetInteger .....	1871
ObjectGetString .....	1873
TextSetFont .....	1875
TextOut .....	1878
TextGetSize .....	1882
<b>26 Технические индикаторы .....</b>	<b>1883</b>
iAC .....	1886
iAD .....	1891
iADX .....	1896
iADXWilder .....	1901
iAlligator .....	1906
iAMA .....	1913
iAO .....	1918
iATR .....	1923
iBearsPower .....	1928
iBands .....	1933
iBullsPower .....	1939
iCCI .....	1944
iChaikin .....	1949
iCustom .....	1954
iDEMA .....	1957
iDeMarker .....	1962
iEnvelopes .....	1967
iForce .....	1973
iFractals .....	1978
iFrAMA .....	1983
iGator .....	1988
iIchimoku .....	1995
iBWMFI .....	2002
iMomentum .....	2007
iMFI .....	2012
iMA .....	2017
iOsMA .....	2022
iMACD .....	2027
iOBV .....	2033

iSAR .....	2038
iRSI .....	2043
iRVI .....	2048
iStdDev .....	2053
iStochastic .....	2058
iTEMA .....	2064
iTriX .....	2069
iWPR .....	2074
iVidyA .....	2079
iVolumes .....	2084
<b>27 Работа с результатами оптимизации.....</b>	<b>2089</b>
FrameFirst .....	2090
FrameFilter .....	2091
FrameNext .....	2092
FrameInputs .....	2093
FrameAdd .....	2094
ParameterGetRange .....	2095
ParameterSetRange .....	2098
<b>28 Работа с событиями.....</b>	<b>2100</b>
EventSetMillisecondTimer .....	2101
EventSetTimer .....	2102
EventKillTimer .....	2103
EventChartCustom .....	2104
<b>29 Работа с OpenCL.....</b>	<b>2110</b>
CLHandleType .....	2112
CLGetInfoInteger .....	2113
CLGetInfoString .....	2116
CLContextCreate .....	2119
CLContextFree .....	2120
CLGetDeviceInfo .....	2121
CLProgramCreate .....	2125
CLProgramFree .....	2130
CLKernelCreate .....	2131
CLKernelFree .....	2132
CLSetKernelArg .....	2133
CLSetKernelArgMem .....	2134
CLSetKernelArgMemLocal .....	2135
CLBufferCreate .....	2136
CLBufferFree .....	2137
CLBufferWrite .....	2138
CLBufferRead .....	2139
CLExecute .....	2140
CLExecutionStatus .....	2142
<b>30 Интеграция .....</b>	<b>2143</b>
MetaTrader для Python .....	2144
MT5Initialize .....	2149
MT5Shutdown .....	2150
MT5TerminalInfo .....	2151
MT5Version .....	2153
MT5WaitForTerminal .....	2155
MT5CopyRatesFrom .....	2156
MT5CopyRatesFromPos .....	2160
MT5CopyRatesRange .....	2163
MT5CopyTicksFrom .....	2166
MT5CopyTicksRange .....	2170
<b>31 Стандартная библиотека.....</b>	<b>2173</b>

<b>Математика .....</b>	<b>2174</b>
<b>Статистика.....</b>	<b>2175</b>
<b>Статистические характеристики.....</b>	<b>2178</b>
<b>MathMean .....</b>	<b>2179</b>
<b>MathVariance.....</b>	<b>2180</b>
<b>MathSkewness.....</b>	<b>2181</b>
<b>MathKurtosis.....</b>	<b>2182</b>
<b>MathMoments.....</b>	<b>2183</b>
<b>MathMedian.....</b>	<b>2184</b>
<b>MathStandardDeviation.....</b>	<b>2185</b>
<b>MathAverageDeviation.....</b>	<b>2186</b>
<b>Нормальное распределение.....</b>	<b>2187</b>
<b>MathProbabilityDensityNormal.....</b>	<b>2191</b>
<b>MathCumulativeDistributionNormal.....</b>	<b>2193</b>
<b>MathQuantileNormal.....</b>	<b>2195</b>
<b>MathRandomNormal.....</b>	<b>2197</b>
<b>MathMomentsNormal.....</b>	<b>2198</b>
<b>Логнормальное распределение.....</b>	<b>2199</b>
<b>MathProbabilityDensityLognormal.....</b>	<b>2203</b>
<b>MathCumulativeDistributionLognormal.....</b>	<b>2205</b>
<b>MathQuantileLognormal.....</b>	<b>2207</b>
<b>MathRandomLognormal.....</b>	<b>2209</b>
<b>MathMomentsLognormal.....</b>	<b>2210</b>
<b>Бета-распределение.....</b>	<b>2211</b>
<b>MathProbabilityDensityBeta .....</b>	<b>2215</b>
<b>MathCumulativeDistributionBeta .....</b>	<b>2217</b>
<b>MathQuantileBeta.....</b>	<b>2219</b>
<b>MathRandomBeta.....</b>	<b>2221</b>
<b>MathMomentsBeta .....</b>	<b>2222</b>
<b>Нецентральное бета-распределение.....</b>	<b>2223</b>
<b>MathProbabilityDensityNoncentralBeta .....</b>	<b>2227</b>
<b>MathCumulativeDistributionNoncentralBeta.....</b>	<b>2229</b>
<b>MathQuantileNoncentralBeta.....</b>	<b>2231</b>
<b>MathRandomNoncentralBeta.....</b>	<b>2233</b>
<b>MathMomentsNoncentralBeta .....</b>	<b>2234</b>
<b>Гамма-распределение.....</b>	<b>2235</b>
<b>MathProbabilityDensityGamma .....</b>	<b>2239</b>
<b>MathCumulativeDistributionGamma .....</b>	<b>2241</b>
<b>MathQuantileGamma.....</b>	<b>2243</b>
<b>MathRandomGamma.....</b>	<b>2245</b>
<b>MathMomentsGamma .....</b>	<b>2246</b>
<b>Распределение хи-квадрат .....</b>	<b>2247</b>
<b>MathProbabilityDensityChiSquare .....</b>	<b>2251</b>
<b>MathCumulativeDistributionChiSquare.....</b>	<b>2253</b>
<b>MathQuantileChiSquare.....</b>	<b>2255</b>
<b>MathRandomChiSquare.....</b>	<b>2257</b>
<b>MathMomentsChiSquare .....</b>	<b>2258</b>
<b>Нецентральное распределение хи-квадрат.....</b>	<b>2259</b>
<b>MathProbabilityDensityNoncentralChiSquare .....</b>	<b>2263</b>
<b>MathCumulativeDistributionNoncentralChiSquare.....</b>	<b>2265</b>
<b>MathQuantileNoncentralChiSquare .....</b>	<b>2267</b>
<b>MathRandomNoncentralChiSquare .....</b>	<b>2269</b>
<b>MathMomentsNoncentralChiSquare .....</b>	<b>2270</b>
<b>Экспоненциальное распределение.....</b>	<b>2271</b>
<b>MathProbabilityDensityExponential .....</b>	<b>2275</b>
<b>MathCumulativeDistributionExponential.....</b>	<b>2277</b>
<b>MathQuantileExponential.....</b>	<b>2279</b>
<b>MathRandomExponential .....</b>	<b>2281</b>

MathMomentsExponential.....	2282
F-распределение.....	2283
MathProbabilityDensityF.....	2287
MathCumulativeDistributionF.....	2289
MathQuantileF.....	2291
MathRandomF.....	2293
MathMomentsF.....	2294
Нецентральное F-распределение .....	2295
MathProbabilityDensityNoncentralF.....	2299
MathCumulativeDistributionNoncentralF.....	2301
MathQuantileNoncentralF.....	2303
MathRandomNoncentralF.....	2305
MathMomentsNoncentralF.....	2306
T-распределение .....	2307
MathProbabilityDensityT.....	2311
MathCumulativeDistributionT.....	2313
MathQuantileT.....	2315
MathRandomT.....	2317
MathMomentsT.....	2318
Нецентральное T-распределение .....	2319
MathProbabilityDensityNoncentralT.....	2323
MathCumulativeDistributionNoncentralT.....	2325
MathQuantileNoncentralT.....	2327
MathRandomNoncentralT.....	2329
MathMomentsNoncentralT.....	2330
Логистическое распределение.....	2331
MathProbabilityDensityLogistic.....	2335
MathCumulativeDistributionLogistic.....	2337
MathQuantileLogistic.....	2339
MathRandomLogistic .....	2341
MathMomentsLogistic.....	2342
Распределение Коши.....	2343
MathProbabilityDensityCauchy.....	2347
MathCumulativeDistributionCauchy.....	2349
MathQuantileCauchy .....	2351
MathRandomCauchy .....	2353
MathMomentsCauchy.....	2354
Равномерное распределение .....	2355
MathProbabilityDensityUniform.....	2359
MathCumulativeDistributionUniform.....	2361
MathQuantileUniform.....	2363
MathRandomUniform.....	2365
MathMomentsUniform.....	2366
Распределение Вейбулла.....	2367
MathProbabilityDensityWeibull.....	2371
MathCumulativeDistributionWeibull.....	2373
MathQuantileWeibull.....	2375
MathRandomWeibull.....	2377
MathMomentsWeibull.....	2378
Биномиальное распределение.....	2379
MathProbabilityDensityBinomial.....	2382
MathCumulativeDistributionBinomial.....	2384
MathQuantileBinomial.....	2386
MathRandomBinomial.....	2388
MathMomentsBinomial.....	2389
Отрицательное биномиальное распределение.....	2390
MathProbabilityDensityNegativeBinomial.....	2393
MathCumulativeDistributionNegativeBinomial.....	2395

MathQuantileNegativeBinomial.....	2397
MathRandomNegativeBinomial.....	2399
MathMomentsNegativeBinomial.....	2400
Геометрическое распределение.....	2401
MathProbabilityDensityGeometric.....	2405
MathCumulativeDistributionGeometric.....	2407
MathQuantileGeometric.....	2409
MathRandomGeometric.....	2411
MathMomentsGeometric.....	2412
Гипергеометрическое распределение.....	2413
MathProbabilityDensityHypergeometric.....	2417
MathCumulativeDistributionHypergeometric.....	2419
MathQuantileHypergeometric.....	2421
MathRandomHypergeometric.....	2423
MathMomentsHypergeometric.....	2424
Распределение Пуассона.....	2425
MathProbabilityDensityPoisson.....	2429
MathCumulativeDistributionPoisson.....	2431
MathQuantilePoisson.....	2433
MathRandomPoisson.....	2435
MathMomentsPoisson.....	2436
Вспомогательные функции.....	2437
MathRandomNonZero.....	2442
MathMoments.....	2443
MathPowInt .....	2444
MathFactorial.....	2445
MathTrunc.....	2446
MathRound.....	2447
MathArctan2.....	2449
MathGamma.....	2451
MathGammaLog.....	2452
MathBeta .....	2453
MathBetaLog.....	2454
MathBetaIncomplete .....	2455
MathGammaIncomplete.....	2456
MathBinomialCoefficient.....	2457
MathBinomialCoefficientLog .....	2458
MathHypergeometric2F2.....	2459
MathSequence.....	2460
MathSequenceByCount.....	2461
MathReplicate .....	2462
MathReverse.....	2463
MathIdentical.....	2464
MathUnique .....	2465
MathQuickSortAscending.....	2466
MathQuickSortDescending .....	2467
MathQuickSort .....	2468
MathOrder .....	2469
MathBitwiseNot .....	2470
MathBitwiseAnd .....	2471
MathBitwiseOr .....	2472
MathBitwiseXor .....	2473
MathBitwiseShiftL .....	2474
MathBitwiseShiftR .....	2475
MathCumulativeSum .....	2476
MathCumulativeProduct .....	2477
MathCumulativeMin .....	2478
MathCumulativeMax .....	2479

MathSin .....	2480
MathCos .....	2481
MathTan .....	2482
MathArcsin.....	2483
MathArccos.....	2484
MathArctan.....	2485
MathSinPi .....	2486
MathCosPi.....	2487
MathTanPi.....	2488
MathAbs .....	2489
MathCeil .....	2490
MathFloor .....	2491
MathSqrt .....	2492
MathExp .....	2493
MathPow .....	2494
MathLog .....	2495
MathLog2 .....	2496
MathLog10.....	2497
MathLog1p.....	2498
MathDifference.....	2499
MathSample.....	2501
MathTukeySummary .....	2504
MathRange .....	2505
MathMin .....	2506
MathMax .....	2507
MathSum .....	2508
MathProduct.....	2509
MathStandardDeviation.....	2510
MathAverageDeviation.....	2511
MathMedian.....	2512
MathMean .....	2513
MathVariance.....	2514
MathSkewness .....	2515
MathKurtosis.....	2516
MathExpm1.....	2517
MathSinh .....	2518
MathCosh .....	2519
MathTanh .....	2520
MathArcsinh.....	2521
MathArccosh.....	2522
MathArctanh.....	2523
MathSignif.....	2524
MathRank .....	2526
MathCorrelationPearson.....	2527
MathCorrelationSpearman.....	2528
MathCorrelationKendall.....	2529
MathQuantile .....	2530
MathProbabilityDensityEmpirical.....	2531
MathCumulativeDistributionEmpirical.....	2532
Нечеткая логика.....	2533
Функции принадлежности.....	2535
CConstantMembershipFunction.....	2537
GetValue .....	2539
CCompositeMembershipFunction.....	2540
CompositionType.....	2542
MembershipFunctions .....	2542
GetValue .....	2542
CDifferenceTwoSigmoidalMembershipFunction.....	2544

A1 .....	2546
A2 .....	2546
C1 .....	2547
C2 .....	2547
GetValue .....	2548
CGeneralizedBellShapedMembershipFunction.....	2549
A .....	2551
B .....	2551
C .....	2552
GetValue .....	2552
CNormalCombinationMembershipFunction.....	2553
B1 .....	2555
B2 .....	2555
Sigma1 .....	2556
Sigma2 .....	2556
GetValue .....	2557
CNormalMembershipFunction.....	2558
B .....	2560
Sigma .....	2560
GetValue .....	2561
CP_ShapedMembershipFunction.....	2562
A .....	2564
B .....	2564
C .....	2565
D .....	2565
GetValue .....	2565
CProductTwoSigmoidalMembershipFunctions.....	2567
A1 .....	2569
A2 .....	2569
C1 .....	2570
C2 .....	2570
GetValue .....	2571
CS_ShapedMembershipFunction.....	2572
A .....	2574
B .....	2574
GetValue .....	2575
CSigmoidalMembershipFunction .....	2576
A .....	2578
C .....	2578
GetValue .....	2579
CTrapezoidMembershipFunction.....	2580
X1 .....	2582
X2 .....	2582
X3 .....	2583
X4 .....	2583
GetValue .....	2584
CTriangularMembershipFunction.....	2585
X1 .....	2587
X2 .....	2587
X3 .....	2588
ToNormalMF .....	2588
GetValue .....	2588
CZ_ShapedMembershipFunction.....	2590
A .....	2592
B .....	2592
GetValue .....	2593
IMembershipFunction.....	2594
GetValue .....	2594

Правила для нечетких систем.....	2595
CMamdaniFuzzyRule.....	2596
Conclusion.....	2597
Weight .....	2597
CSugenoFuzzyRule.....	2598
Conclusion.....	2599
CSingleCondition.....	2600
Not .....	2600
Term .....	2601
Var .....	2601
CConditions.....	2603
ConditionsList.....	2603
Not .....	2604
Op .....	2604
CGenericFuzzyRule.....	2606
Conclusion.....	2606
Condition .....	2607
CreateCondition.....	2607
Переменные для нечетких систем.....	2609
CFuzzyVariable.....	2610
AddTerm .....	2611
GetTermByName.....	2611
Max .....	2611
Min .....	2612
Terms .....	2612
Values .....	2612
CSugenoVariable.....	2614
Functions .....	2614
GetFuncByName.....	2615
Values .....	2615
Термы нечетких переменных.....	2616
MembershipFunction.....	2617
Нечеткие системы.....	2618
Система Мамдани.....	2619
AggregationMethod.....	2619
Calculate .....	2620
DefuzzificationMethod.....	2620
EmptyRule .....	2620
ImplicationMethod.....	2620
Output .....	2621
OutputByName.....	2621
ParseRule .....	2621
Rules .....	2622
Система Сугено.....	2623
Calculate .....	2623
CreateSugenoFunction.....	2624
EmptyRule .....	2625
Output .....	2625
OutputByName.....	2625
ParseRule .....	2625
Rules .....	2626
OpenCL .....	2627
BufferCreate.....	2629
BufferFree.....	2630
BufferFromArray.....	2631
BufferRead.....	2632
BufferWrite.....	2633
Execute.....	2634

GetContext .....	2635
GetKernel.....	2636
GetKernelName.....	2637
GetProgram.....	2638
Initialize.....	2639
KernelCreate.....	2640
KernelFree.....	2641
SetArgument.....	2642
SetArgumentBuffer.....	2643
SetArgumentLocalMemory.....	2644
SetBuffersCount.....	2645
SetKernelsCount.....	2646
Shutdown.....	2647
SupportDouble.....	2648
<b>Базовый класс CObject .....</b>	<b>2649</b>
Prev .....	2651
Prev .....	2652
Next .....	2653
Next .....	2654
Compare.....	2655
Save .....	2657
Load .....	2659
Type .....	2661
<b>Коллекции данных .....</b>	<b>2662</b>
CArray .....	2664
Step .....	2666
Step .....	2667
Total .....	2668
Available .....	2669
Max .....	2670
IsSorted .....	2671
SortMode .....	2672
Clear .....	2673
Sort .....	2674
Save .....	2675
Load .....	2677
CArrayChar .....	2679
Reserve .....	2682
Resize .....	2683
Shutdown .....	2684
Add .....	2685
AddArray .....	2686
AddArray .....	2687
Insert .....	2689
InsertArray.....	2690
InsertArray.....	2692
AssignArray.....	2694
AssignArray.....	2695
Update .....	2697
Shift .....	2698
Delete .....	2699
DeleteRange.....	2700
At .....	2701
CompareArray .....	2703
CompareArray .....	2704
InsertSort .....	2705
Search .....	2706
SearchGreat.....	2707

SearchLess .....	2708
SearchGreatOrEqual .....	2709
SearchLessOrEqual .....	2710
SearchFirst .....	2711
SearchLast .....	2712
SearchLinear .....	2713
Save .....	2714
Load .....	2716
Type .....	2718
CArrayShort .....	2719
Reserve .....	2722
Resize .....	2723
Shutdown .....	2724
Add .....	2725
AddArray .....	2726
AddArray .....	2727
Insert .....	2729
InsertArray .....	2730
InsertArray .....	2731
AssignArray .....	2733
AssignArray .....	2734
Update .....	2736
Shift .....	2737
Delete .....	2738
DeleteRange .....	2739
At .....	2740
CompareArray .....	2742
CompareArray .....	2743
InsertSort .....	2744
Search .....	2745
SearchGreat .....	2746
SearchLess .....	2747
SearchGreatOrEqual .....	2748
SearchLessOrEqual .....	2749
SearchFirst .....	2750
SearchLast .....	2751
SearchLinear .....	2752
Save .....	2753
Load .....	2755
Type .....	2757
CArrayInt .....	2758
Reserve .....	2761
Resize .....	2762
Shutdown .....	2763
Add .....	2764
AddArray .....	2765
AddArray .....	2766
Insert .....	2768
InsertArray .....	2769
InsertArray .....	2770
AssignArray .....	2772
AssignArray .....	2773
Update .....	2775
Shift .....	2776
Delete .....	2777
DeleteRange .....	2778
At .....	2779
CompareArray .....	2781

CompareArray .....	2782
InsertSort .....	2783
Search .....	2784
SearchGreat.....	2785
SearchLess .....	2786
SearchGreaterOrEqual.....	2787
SearchLessOrEqual.....	2788
SearchFirst.....	2789
SearchLast .....	2790
SearchLinear.....	2791
Save .....	2792
Load .....	2794
Type .....	2796
<b>CArrayLong .....</b>	<b>2797</b>
Reserve .....	2800
Resize .....	2801
Shutdown .....	2802
Add .....	2803
AddArray .....	2804
AddArray .....	2805
Insert .....	2807
InsertArray.....	2808
InsertArray.....	2809
AssignArray.....	2811
AssignArray.....	2812
Update .....	2814
Shift .....	2815
Delete .....	2816
DeleteRange.....	2817
At .....	2818
CompareArray.....	2820
CompareArray .....	2821
InsertSort .....	2822
Search .....	2823
SearchGreat.....	2824
SearchLess .....	2825
SearchGreaterOrEqual.....	2826
SearchLessOrEqual.....	2827
SearchFirst.....	2828
SearchLast .....	2829
SearchLinear.....	2830
Save .....	2831
Load .....	2833
Type .....	2835
<b>CArrayFloat .....</b>	<b>2836</b>
Delta .....	2839
Reserve .....	2840
Resize .....	2841
Shutdown .....	2842
Add .....	2843
AddArray .....	2844
AddArray .....	2845
Insert .....	2847
InsertArray.....	2848
InsertArray.....	2849
AssignArray.....	2851
AssignArray.....	2852
Update .....	2854

Shift .....	2855
Delete .....	2856
DeleteRange.....	2857
At .....	2858
CompareArray .....	2860
CompareArray.....	2861
InsertSort .....	2862
Search .....	2863
SearchGreat.....	2864
SearchLess.....	2865
SearchGreatOrEqual.....	2866
SearchLessOrEqual.....	2867
SearchFirst.....	2868
SearchLast .....	2869
SearchLinear.....	2870
Save .....	2871
Load .....	2873
Type .....	2875
CArrayDouble.....	2876
Delta .....	2879
Reserve .....	2880
Resize .....	2881
Shutdown .....	2882
Add .....	2883
AddArray .....	2884
AddArray .....	2885
Insert .....	2887
InsertArray.....	2888
InsertArray.....	2889
AssignArray.....	2891
AssignArray.....	2892
Update .....	2894
Shift .....	2895
Delete .....	2896
DeleteRange.....	2897
At .....	2898
CompareArray .....	2900
CompareArray.....	2901
Minimum .....	2902
Maximum .....	2903
InsertSort .....	2904
Search .....	2905
SearchGreat.....	2906
SearchLess.....	2907
SearchGreatOrEqual.....	2908
SearchLessOrEqual.....	2909
SearchFirst.....	2910
SearchLast .....	2911
SearchLinear.....	2912
Save .....	2913
Load .....	2915
Type .....	2917
CArrayList.....	2918
Reserve .....	2921
Resize .....	2922
Shutdown .....	2923
Add .....	2924
AddArray .....	2925

AddArray .....	2926
Insert .....	2928
InsertArray.....	2929
InsertArray.....	2930
AssignArray.....	2932
AssignArray.....	2933
Update .....	2935
Shift .....	2936
Delete .....	2937
DeleteRange.....	2938
At .....	2939
CompareArray .....	2941
CompareArray .....	2942
InsertSort .....	2943
Search .....	2944
SearchGreat.....	2945
SearchLess.....	2946
SearchGreatOrEqual.....	2947
SearchLessOrEqual.....	2948
SearchFirst.....	2949
SearchLast .....	2950
SearchLinear.....	2951
Save .....	2952
Load .....	2954
Type .....	2956
CArrayObj.....	2957
FreeMode .....	2962
FreeMode .....	2963
Reserve .....	2965
Resize .....	2966
Clear .....	2968
Shutdown .....	2969
CreateElement.....	2970
Add .....	2972
AddArray .....	2974
Insert .....	2977
InsertArray.....	2979
AssignArray.....	2981
Update .....	2983
Shift .....	2985
Detach .....	2986
Delete .....	2988
DeleteRange.....	2989
At .....	2991
CompareArray .....	2993
InsertSort .....	2994
Search .....	2995
SearchGreat.....	2997
SearchLess.....	2999
SearchGreatOrEqual.....	3001
SearchLessOrEqual.....	3003
SearchFirst.....	3005
SearchLast .....	3007
Save .....	3009
Load .....	3011
Type .....	3013
CList .....	3014
FreeMode .....	3017

FreeMode .....	3018
Total .....	3020
IsSorted .....	3021
SortMode .....	3022
CreateElement.....	3023
Add .....	3024
Insert .....	3026
DetachCurrent.....	3028
DeleteCurrent.....	3029
Delete .....	3030
Clear .....	3031
IndexOf .....	3032
GetNodeAtIndex.....	3033
GetFirstNode.....	3034
GetPrevNode.....	3035
GetCurrentNode.....	3036
GetNextNode.....	3037
GetLastNode .....	3038
Sort .....	3039
MoveToIndex.....	3040
Exchange .....	3041
CompareList.....	3042
Search .....	3043
Save .....	3045
Load .....	3047
Type .....	3049
CTreeNode.....	3050
Owner .....	3055
Left .....	3056
Right .....	3057
Balance .....	3058
BalanceL .....	3059
BalanceR .....	3060
CreateSample.....	3061
RefreshBalance.....	3062
GetNext .....	3063
SaveNode .....	3064
LoadNode .....	3065
Type .....	3066
CTree .....	3067
Root .....	3073
CreateElement.....	3074
Insert .....	3075
Detach .....	3076
Delete .....	3077
Clear .....	3078
Find .....	3079
Save .....	3080
Load .....	3081
Type .....	3082
<b>Шаблонные коллекции данных .....</b>	<b>3083</b>
ICollection<T> .....	3086
Add .....	3087
Count .....	3088
Contains .....	3089
CopyTo .....	3090
Clear .....	3091
Remove .....	3092

IEqualityComparable<T> .....	3093
Equals .....	3094
HashCode .....	3095
IComparable<T> .....	3096
Compare .....	3097
IComparer<T> .....	3098
Compare .....	3099
IEqualityComparer<T> .....	3100
Equals .....	3101
HashCode .....	3102
IList<T> .....	3103
TryGetValue.....	3104
TrySetValue.....	3105
Insert .....	3106
IndexOf .....	3107
LastIndexOf.....	3108
RemoveAt .....	3109
IMap< TKey , TValue >.....	3110
Add .....	3111
Contains .....	3112
Remove .....	3113
TryGetValue.....	3114
TrySetValue.....	3115
CopyTo .....	3116
ISet<T> .....	3117
ExceptWith.....	3119
IntersectWith.....	3120
SymmetricExceptWith.....	3121
UnionWith .....	3122
IsProperSubsetOf .....	3123
IsProperSupersetOf.....	3124
IsSubsetOf .....	3125
IsSupersetOf .....	3126
Overlaps .....	3127
SetEquals .....	3128
CDefaultComparer<T> .....	3129
Compare .....	3130
CDefaultEqualityComparer<T> .....	3131
Equals .....	3132
HashCode .....	3133
CRedBlackTreeNode<T> .....	3134
Value .....	3135
Parent .....	3136
Left .....	3137
Right .....	3138
Color .....	3139
IsLeaf .....	3140
CreateEmptyNode.....	3141
CLinkedListNode<T> .....	3142
List .....	3143
Next .....	3144
Previous .....	3145
Value .....	3146
CKeyValuePair< TKey , TValue > .....	3147
Key .....	3148
Value .....	3149
Clone .....	3150
Compare .....	3151

Equals .....	3152
HashCode .....	3153
CArrayList<T>.....	3154
Capacity .....	3156
Count .....	3157
Contains .....	3158
TrimExcess.....	3159
TryGetValue.....	3160
TrySetValue.....	3161
Add .....	3162
AddRange .....	3163
Insert .....	3164
InsertRange.....	3165
CopyTo .....	3166
BinarySearch.....	3167
IndexOf .....	3168
LastIndexOf.....	3169
Clear .....	3170
Remove .....	3171
RemoveAt .....	3172
RemoveRange.....	3173
Reverse .....	3174
Sort .....	3175
CHashMap< TKey, TValue > .....	3176
Add .....	3178
Count .....	3179
Comparer .....	3180
Contains .....	3181
ContainsKey.....	3182
ContainsValue.....	3183
CopyTo .....	3184
Clear .....	3185
Remove .....	3186
TryGetValue.....	3187
TrySetValue.....	3188
CHashSet<T>.....	3189
Add .....	3191
Count .....	3192
Contains .....	3193
Comparer .....	3194
TrimExcess .....	3195
CopyTo .....	3196
Clear .....	3197
Remove .....	3198
ExceptWith.....	3199
IntersectWith.....	3200
SymmetricExceptWith.....	3201
UnionWith .....	3202
IsProperSubsetOf.....	3203
IsProperSupersetOf.....	3204
IsSubsetOf.....	3205
IsSupersetOf.....	3206
Overlaps .....	3207
SetEquals .....	3208
CLinkedList<T>.....	3209
Add .....	3211
AddAfter .....	3212
AddBefore .....	3213

AddFirst .....	3214
AddLast .....	3215
Count .....	3216
Head .....	3217
First .....	3218
Last .....	3219
Contains .....	3220
CopyTo .....	3221
Clear .....	3222
Remove .....	3223
RemoveFirst .....	3224
RemoveLast .....	3225
Find .....	3226
FindLast .....	3227
CQueue<T>.....	3228
Add .....	3229
Enqueue .....	3230
Count .....	3231
Contains .....	3232
TrimExcess .....	3233
CopyTo .....	3234
Clear .....	3235
Remove .....	3236
Dequeue .....	3237
Peek .....	3238
CRedBlackTree<T> .....	3239
Add .....	3241
Count .....	3242
Root .....	3243
Contains .....	3244
Comparer .....	3245
TryGetMin .....	3246
TryGetMax .....	3247
CopyTo .....	3248
Clear .....	3249
Remove .....	3250
RemoveMin .....	3251
RemoveMax .....	3252
Find .....	3253
FindMin .....	3254
FindMax .....	3255
CSortedMap< TKey, TValue > .....	3256
Add .....	3258
Count .....	3259
Comparer .....	3260
Contains .....	3261
ContainsKey .....	3262
ContainsValue .....	3263
CopyTo .....	3264
Clear .....	3265
Remove .....	3266
TryGetValue .....	3267
TrySetValue .....	3268
CSortedSet< T >.....	3269
Add .....	3271
Count .....	3272
Contains .....	3273
Comparer .....	3274

TryGetMin .....	3275
TryGetMax .....	3276
CopyTo .....	3277
Clear .....	3278
Remove .....	3279
ExceptWith .....	3280
IntersectWith .....	3281
SymmetricExceptWith .....	3282
UnionWith .....	3283
IsProperSubsetOf .....	3284
IsProperSupersetOf .....	3285
IsSubsetOf .....	3286
IsSupersetOf .....	3287
Overlaps .....	3288
SetEquals .....	3289
GetViewBetween .....	3290
GetReverse .....	3291
CStack<T> .....	3292
Add .....	3293
Count .....	3294
Contains .....	3295
TrimExcess .....	3296
CopyTo .....	3297
Clear .....	3298
Remove .....	3299
Push .....	3300
Peek .....	3301
Pop .....	3302
ArrayBinarySearch<T> .....	3303
ArrayIndexOf<T> .....	3304
ArrayLastIndexOf<T> .....	3305
ArrayReverse<T> .....	3306
Compare .....	3307
Equals<T> .....	3310
GetHashCode .....	3311
Файлы .....	3314
CFile .....	3315
Handle .....	3317
Filename .....	3318
Flags .....	3319
SetUnicode .....	3320
SetCommon .....	3321
Open .....	3322
Close .....	3323
Delete .....	3324
IsExist .....	3325
Copy .....	3326
Move .....	3327
Size .....	3328
Tell .....	3329
Seek .....	3330
Flush .....	3331
IsEnding .....	3332
IsLineEnding .....	3333
FolderCreate .....	3334
FolderDelete .....	3335
FolderClean .....	3336
FileFindFirst .....	3337

FileFindNext.....	3338
FileFindClose .....	3339
CFileBin.....	3340
Open .....	3342
WriteChar .....	3343
WriteShort .....	3344
WriteInteger.....	3345
WriteLong .....	3346
WriteFloat.....	3347
WriteDouble.....	3348
WriteString.....	3349
WriteCharArray.....	3350
WriteShortArray .....	3351
WriteIntegerArray.....	3352
WriteLongArray .....	3353
WriteFloatArray.....	3354
WriteDoubleArray.....	3355
WriteObject .....	3356
ReadChar .....	3357
ReadShort .....	3358
ReadInteger.....	3359
ReadLong .....	3360
ReadFloat .....	3361
ReadDouble .....	3362
ReadString.....	3363
ReadCharArray.....	3364
ReadShortArray .....	3365
ReadIntegerArray.....	3366
ReadLongArray .....	3367
ReadFloatArray .....	3368
ReadDoubleArray .....	3369
ReadObject .....	3370
CFileTxt.....	3371
Open .....	3372
WriteString.....	3373
ReadString.....	3374
Строки .....	3375
CString.....	3376
Str .....	3378
Len .....	3379
Copy .....	3380
Fill .....	3381
Assign .....	3382
Append .....	3383
Insert .....	3384
Compare .....	3385
CompareNoCase.....	3386
Left .....	3387
Right .....	3388
Mid .....	3389
Trim .....	3390
TrimLeft .....	3391
TrimRight .....	3392
Clear .....	3393
ToUpper .....	3394
ToLower .....	3395
Reverse .....	3396
Find .....	3397

FindRev .....	3398
Remove .....	3399
Replace .....	3400
<b>Графические объекты.....</b>	<b>3401</b>
Базовый класс CChartObject .....	3402
ChartId .....	3405
Window .....	3406
Name .....	3407
NumPoints .....	3408
Attach .....	3409
SetPoint .....	3410
Delete .....	3411
Detach .....	3412
ShiftObject.....	3413
ShiftPoint .....	3414
Time .....	3415
Price .....	3417
Color .....	3419
Style .....	3420
Width .....	3421
Background.....	3422
Selected .....	3423
Selectable .....	3424
Description.....	3425
Tooltip .....	3426
Timeframes.....	3427
Z_Order .....	3428
CreateTime.....	3429
LevelsCount.....	3430
LevelColor .....	3431
LevelStyle .....	3433
LevelWidth.....	3435
LevelValue.....	3437
LevelDescription .....	3439
GetInteger .....	3441
SetInteger .....	3443
GetDouble .....	3445
SetDouble .....	3447
GetString .....	3449
SetString .....	3451
Save .....	3453
Load .....	3454
Type .....	3455
Объекты "Линии".....	3456
CChartObjectVLine.....	3457
Create .....	3458
Type .....	3459
CChartObjectHLine.....	3460
Create .....	3461
Type .....	3462
CChartObjectTrend.....	3463
Create .....	3465
RayLeft .....	3466
RayRight .....	3467
Save .....	3468
Load .....	3469
Type .....	3470
CChartObjectTrendByAngle.....	3471

Create .....	3473
Angle .....	3474
Type .....	3475
CChartObjectCycles.....	3476
Create .....	3477
Type .....	3478
Объекты "Каналы".....	3479
CChartObjectChannel.....	3480
Create .....	3482
Type .....	3483
CChartObjectRegression.....	3484
Create .....	3486
Type .....	3487
CChartObjectStdDevChannel.....	3488
Create .....	3490
Deviations.....	3491
Save .....	3492
Load .....	3493
Type .....	3494
CChartObjectPitchfork.....	3495
Create .....	3497
Type .....	3498
Инструменты Ганна.....	3499
CChartObjectGannLine.....	3500
Create .....	3502
PipsPerBar.....	3503
Save .....	3504
Load .....	3505
Type .....	3506
CChartObjectGannFan.....	3507
Create .....	3509
PipsPerBar.....	3510
Downtrend.....	3511
Save .....	3512
Load .....	3513
Type .....	3514
CChartObjectGannGrid.....	3515
Create .....	3517
PipsPerBar.....	3518
Downtrend.....	3519
Save .....	3520
Load .....	3521
Type .....	3522
Инструменты Фибоначчи.....	3523
CChartObjectFibo.....	3524
Create .....	3526
Type .....	3527
CChartObjectFiboTimes.....	3528
Create .....	3529
Type .....	3530
CChartObjectFiboFan.....	3531
Create .....	3532
Type .....	3533
CChartObjectFiboArc.....	3534
Create .....	3536
Scale .....	3537
Ellipse .....	3538
Save .....	3539

Load .....	3540
Type .....	3541
CChartObjectFiboChannel.....	3542
Create .....	3544
Type .....	3545
CChartObjectFiboExpansion.....	3546
Create .....	3548
Type .....	3549
Инструменты Эллиотта.....	3550
CChartObjectElliottWave3.....	3551
Create .....	3553
Degree .....	3554
Lines .....	3555
Save .....	3556
Load .....	3557
Type .....	3558
CChartObjectElliottWave5.....	3559
Create .....	3561
Type .....	3563
Объекты "Фигуры".....	3564
CChartObjectRectangle.....	3565
Create .....	3566
Type .....	3567
CChartObjectTriangle.....	3568
Create .....	3569
Type .....	3570
CChartObjectEllipse.....	3571
Create .....	3572
Type .....	3573
Объекты "Стрелки".....	3574
CChartObjectArrow.....	3575
Create .....	3577
ArrowCode.....	3579
Anchor .....	3581
Save .....	3583
Load .....	3584
Type .....	3585
Стрелки с фиксированным кодом.....	3586
Create .....	3588
ArrowCode.....	3590
Type .....	3592
Элементы управления.....	3593
CChartObjectText.....	3594
Create .....	3596
Angle .....	3597
Font .....	3598
FontSize .....	3599
Anchor .....	3600
Save .....	3601
Load .....	3602
Type .....	3603
CChartObjectLabel.....	3604
Create .....	3606
X_Distance.....	3607
Y_Distance.....	3608
X_Size .....	3609
Y_Size .....	3610
Corner .....	3611

Time .....	3612
Price .....	3613
Save .....	3614
Load .....	3615
Type .....	3616
CChartObjectEdit.....	3617
Create .....	3619
TextAlign .....	3620
X_Size .....	3621
Y_Size .....	3622
BackColor .....	3623
BorderColor .....	3624
ReadOnly .....	3625
Angle .....	3626
Save .....	3627
Load .....	3628
Type .....	3629
CChartObjectButton.....	3630
State .....	3632
Save .....	3633
Load .....	3634
Type .....	3635
CChartObjectSubChart.....	3636
Create .....	3638
X_Distance.....	3639
Y_Distance.....	3640
Corner .....	3641
X_Size .....	3642
Y_Size .....	3643
Symbol .....	3644
Period .....	3645
Scale .....	3646
DateScale .....	3647
PriceScale .....	3648
Time .....	3649
Price .....	3650
Save .....	3651
Load .....	3652
Type .....	3653
CChartObjectBitmap.....	3654
Create .....	3656
BmpFile .....	3657
X_Offset .....	3658
Y_Offset .....	3659
Save .....	3660
Load .....	3661
Type .....	3662
CChartObjectBmpLabel.....	3663
Create .....	3665
X_Distance.....	3666
Y_Distance.....	3667
X_Offset .....	3668
Y_Offset .....	3669
Corner .....	3670
X_Size .....	3671
Y_Size .....	3672
BmpFileOn .....	3673
BmpFileOff .....	3674

State .....	3675
Time .....	3676
Price .....	3677
Save .....	3678
Load .....	3679
Type .....	3680
CChartObjectRectLabel.....	3681
Create .....	3683
X_Size .....	3684
Y_Size .....	3685
BackColor .....	3686
Angle .....	3687
BorderType.....	3688
Save .....	3689
Load .....	3690
Type .....	3691
<b>Пользовательская графика .....</b>	<b>3692</b>
CCanvas.....	3693
Attach .....	3697
Arc .....	3698
Pie .....	3701
FillPolygon .....	3705
FillEllipse .....	3706
GetDefaultColor .....	3707
ChartObjectName.....	3708
Circle .....	3709
CircleAA .....	3710
CircleWu .....	3711
Create .....	3712
CreateBitmap.....	3713
CreateBitmapLabel.....	3715
Destroy .....	3717
Ellipse .....	3718
EllipseAA .....	3719
EllipseWu .....	3720
Erase .....	3721
Fill .....	3722
FillCircle .....	3723
FillRectangle.....	3724
FillTriangle.....	3725
FontAngleGet.....	3726
FontAngleSet .....	3727
FontFlagsGet.....	3728
FontFlagsSet .....	3729
FontGet .....	3730
FontNameGet .....	3731
FontNameSet.....	3732
FontSet .....	3733
FontSizeGet .....	3734
FontSizeSet.....	3735
Height .....	3736
Line .....	3737
LineAA .....	3738
LineWu .....	3739
LineHorizontal.....	3740
LineVertical.....	3741
LineStyleSet .....	3742
LineThick .....	3743

LineThickVertical.....	3744
LineThickHorizontal.....	3745
LoadFromFile.....	3746
PixelGet .....	3747
PixelSet .....	3748
PixelSetAA .....	3749
Polygon .....	3750
PolygonAA .....	3751
PolygonWu.....	3752
PolygonThick .....	3753
PolygonSmooth.....	3754
Polyline .....	3755
PolylineSmooth.....	3756
PolylineThick .....	3757
PolylineWu.....	3758
PolylineAA .....	3759
Rectangle .....	3760
Resize .....	3761
ResourceName.....	3762
TextHeight .....	3763
TextOut .....	3764
TextSize .....	3765
TextWidth.....	3766
TransparentLevelSet.....	3767
Triangle .....	3768
TriangleAA.....	3769
TriangleWu.....	3770
Update .....	3771
Width .....	3772
<b>CChartCanvas.....</b>	<b>3773</b>
ColorBackground.....	3777
ColorBorder.....	3778
ColorText .....	3779
ColorGrid .....	3780
MaxData .....	3781
MaxDescrLen.....	3782
ShowFlags .....	3783
IsShowLegend.....	3784
IsShowScaleLeft .....	3785
IsShowScaleRight .....	3786
IsShowScaleTop.....	3787
IsShowScaleBottom.....	3788
IsShowGrid.....	3789
IsShowDescriptors .....	3790
IsShowPercent.....	3791
VScaleMin .....	3792
VScaleMax .....	3793
NumGrid .....	3794
DataOffset.....	3795
DataTotal .....	3796
DrawDescriptors .....	3797
DrawData .....	3798
Create .....	3799
AllowedShowFlags.....	3800
ShowLegend.....	3801
ShowScaleLeft.....	3802
ShowScaleRight .....	3803
ShowScaleTop.....	3804

ShowScaleBottom.....	3805
ShowGrid .....	3806
ShowDescriptors.....	3807
ShowValue .....	3808
ShowPercent .....	3809
LegendAlignment.....	3810
Accumulative.....	3811
VScaleParams .....	3812
DescriptorUpdate.....	3813
ColorUpdate.....	3814
ValuesCheck.....	3815
Redraw .....	3816
DrawBackground.....	3817
DrawLegend.....	3818
DrawLegendVertical.....	3819
DrawLegendHorizontal.....	3820
CalcScales .....	3821
DrawScales .....	3822
DrawScaleLeft .....	3823
DrawScaleRight .....	3824
DrawScaleTop.....	3825
DrawScaleBottom.....	3826
DrawGrid .....	3827
DrawChart.....	3828
CHistogramChart.....	3829
Gradient .....	3834
BarGap .....	3835
BarMinSize.....	3836
BarBorder .....	3837
Create .....	3838
SeriesAdd .....	3839
SeriesInsert.....	3840
SeriesUpdate.....	3841
SeriesDelete.....	3842
ValueUpdate.....	3843
DrawData .....	3844
DrawBar .....	3845
GradientBrush.....	3846
CLineChart.....	3847
Filled .....	3851
Create .....	3852
SeriesAdd .....	3853
SeriesInsert.....	3854
SeriesUpdate.....	3855
SeriesDelete.....	3856
ValueUpdate.....	3857
DrawChart.....	3858
DrawData .....	3859
CalcArea .....	3860
CPieChart.....	3861
Create .....	3866
SeriesSet .....	3867
ValueAdd .....	3868
ValueInsert.....	3869
ValueUpdate.....	3870
ValueDelete.....	3871
DrawChart.....	3872
DrawPie .....	3873

LabelMake .....	3874
<b>Ценовые графики .....</b>	<b>3875</b>
ChartID .....	3880
Mode .....	3881
Foreground .....	3882
Shift .....	3883
ShiftSize .....	3884
AutoScroll .....	3885
Scale .....	3886
ScaleFix .....	3887
ScaleFix_11 .....	3888
FixedMax .....	3889
FixedMin .....	3890
ScalePPB .....	3891
PointsPerBar .....	3892
ShowOHLC .....	3893
ShowLineBid .....	3894
ShowLineAsk .....	3895
ShowLastLine .....	3896
ShowPeriodSep .....	3897
ShowGrid .....	3898
ShowVolumes .....	3899
ShowObjectDescr .....	3900
ShowDateScale .....	3901
ShowPriceScale .....	3902
ColorBackground .....	3903
ColorForeground .....	3904
ColorGrid .....	3905
ColorBarUp .....	3906
ColorBarDown .....	3907
ColorCandleBull .....	3908
ColorCandleBear .....	3909
ColorChartLine .....	3910
ColorVolumes .....	3911
ColorLineBid .....	3912
ColorLineAsk .....	3913
ColorLineLast .....	3914
ColorStopLevels .....	3915
VisibleBars .....	3916
WindowsTotal .....	3917
WindowIsVisible .....	3918
WindowHandle .....	3919
FirstVisibleBar .....	3920
WidthInBars .....	3921
WidthInPixels .....	3922
HeightInPixels .....	3923
PriceMin .....	3924
PriceMax .....	3925
Attach .....	3926
FirstChart .....	3927
NextChart .....	3928
Open .....	3929
Detach .....	3930
Close .....	3931
BringToTop .....	3932
EventObjectCreate .....	3933
EventObjectDelete .....	3934
IndicatorAdd .....	3935

IndicatorDelete.....	3936
IndicatorsTotal.....	3937
IndicatorName.....	3938
Navigate.....	3939
Symbol .....	3940
Period .....	3941
Redraw.....	3942
GetInteger.....	3943
SetInteger.....	3944
GetDouble.....	3945
SetDouble.....	3946
GetString.....	3947
SetString.....	3948
SetSymbolPeriod.....	3949
ApplyTemplate.....	3950
ScreenShot.....	3951
WindowOnDropped.....	3952
PriceOnDropped.....	3953
TimeOnDropped.....	3954
XOnDropped.....	3955
YOnDropped.....	3956
Save .....	3957
Load .....	3958
Type .....	3959
<b>Научные графики .....</b>	<b>3960</b>
GraphPlot.....	3961
CAxis .....	3965
AutoScale .....	3967
Min .....	3968
Max .....	3969
Step .....	3970
Name .....	3971
Color .....	3972
ValuesSize .....	3973
ValuesWidth.....	3974
ValuesFormat.....	3975
ValuesDateTimeMode.....	3976
ValuesFunctionFormat .....	3977
ValuesFunctionFormatCBData .....	3979
NameSize .....	3980
ZeroLever .....	3981
DefaultStep.....	3982
MaxLabels .....	3983
MinGrace .....	3984
MaxGrace .....	3985
SelectAxisScale.....	3986
CColorGenerator.....	3987
Next .....	3988
Reset .....	3989
CCurve.....	3990
Type .....	3993
Name .....	3994
Color .....	3995
XMax .....	3996
XMin .....	3997
YMax .....	3998
YMin .....	3999
Size .....	4000

PointSize .....	4001
PointsFill .....	4002
PointsColor .....	4003
GetX .....	4004
GetY .....	4005
LineStyle .....	4006
LinesIsSmooth.....	4007
LinesSmoothTension.....	4008
LinesSmoothStep .....	4009
LinesEndStyle.....	4010
LinesWidth .....	4011
HistogramWidth.....	4013
CustomPlotCBData.....	4014
CustomPlotFunction.....	4015
PointsType.....	4019
StepsDimension.....	4020
TrendLineCoefficients.....	4021
TrendLineColor .....	4022
TrendLineVisible.....	4023
Update .....	4025
Visible .....	4027
<b>CGraphic.....</b>	<b>4028</b>
Create .....	4031
Destroy .....	4032
Update .....	4033
ChartObjectName.....	4034
ResourceName.....	4035
XAxis .....	4036
YAxis .....	4037
GapSize .....	4038
BackgroundColor .....	4039
BackgroundMain.....	4040
BackgroundMainSize.....	4041
BackgroundMainColor .....	4042
BackgroundSub.....	4043
BackgroundSubSize .....	4044
BackgroundSubColor .....	4045
GridLineColor .....	4046
GridBackgroundColor .....	4047
GridCircleRadius .....	4048
GridCircleColor .....	4049
GridHasCircle .....	4050
GridAxisLineColor .....	4051
HistoryNameWidth .....	4052
HistoryNameSize .....	4053
HistorySymbolSize .....	4054
TextAdd .....	4055
LineAdd .....	4056
CurveAdd .....	4057
CurvePlot .....	4060
CurvePlotAll.....	4061
CurvesTotal.....	4062
CurveGetByIndex .....	4063
CurveGetByName .....	4064
CurveRemoveByIndex .....	4065
CurveRemoveByName .....	4066
CurvesTotal.....	4067
MarksToAxisAdd .....	4068

MajorMarkSize.....	4069
FontSet .....	4070
FontGet .....	4071
Attach .....	4072
CalculateMaxMinValues.....	4073
Height .....	4074
IndentDown.....	4075
IndentLeft.....	4076
IndentRight.....	4077
IndentUp .....	4078
Redraw .....	4079
ResetParameters.....	4080
ScaleX .....	4081
ScaleY .....	4082
SetDefaultParameters.....	4083
Width .....	4084
<b>Индикаторы .....</b>	<b>4085</b>
Базовые классы.....	4086
CSpreadBuffer .....	4087
Size .....	4089
SetSymbolPeriod.....	4090
At .....	4091
Refresh .....	4092
RefreshCurrent.....	4093
CTimeBuffer .....	4094
Size .....	4096
SetSymbolPeriod.....	4097
At .....	4098
Refresh .....	4099
RefreshCurrent.....	4100
CTickVolumeBuffer.....	4101
Size .....	4103
SetSymbolPeriod.....	4104
At .....	4105
Refresh .....	4106
RefreshCurrent.....	4107
CRealVolumeBuffer.....	4108
Size .....	4110
SetSymbolPeriod.....	4111
At .....	4112
Refresh .....	4113
RefreshCurrent.....	4114
CDoubleBuffer.....	4115
Size .....	4117
SetSymbolPeriod.....	4118
At .....	4119
Refresh .....	4120
RefreshCurrent.....	4121
COpenBuffer.....	4122
Refresh .....	4123
RefreshCurrent.....	4124
CHighBuffer .....	4125
Refresh .....	4126
RefreshCurrent.....	4127
CLowBuffer.....	4128
Refresh .....	4129
RefreshCurrent.....	4130
CCloseBuffer .....	4131

Refresh .....	4132
RefreshCurrent.....	4133
CIIndicatorBuffer.....	4134
Offset .....	4136
Name .....	4137
At .....	4138
Refresh .....	4139
RefreshCurrent.....	4140
CSeries .....	4141
Name .....	4143
BuffersTotal.....	4144
Timeframe.....	4145
Symbol .....	4146
Period .....	4147
RefreshCurrent.....	4148
BufferSize .....	4149
BufferResize.....	4150
Refresh .....	4151
PeriodDescription.....	4152
CPriceSeries.....	4153
BufferResize.....	4155
GetData .....	4156
Refresh .....	4157
MinIndex .....	4158
MinValue .....	4159
MaxIndex .....	4160
MaxValue .....	4161
CIIndicator.....	4162
Handle .....	4164
Status .....	4165
FullRelease.....	4166
Create .....	4167
BufferResize.....	4168
BarsCalculated.....	4169
GetData .....	4170
Refresh .....	4173
Minimum .....	4174
MinValue .....	4175
Maximum .....	4176
MaxValue .....	4177
MethodDescription.....	4178
PriceDescription.....	4179
VolumeDescription.....	4180
AddToChart.....	4181
DeleteFromChart.....	4182
CIndicators.....	4183
Create .....	4184
Refresh .....	4185
Таймсерии.....	4186
CiSpread .....	4187
Create .....	4189
BufferResize.....	4190
GetData .....	4191
Refresh .....	4193
CiTime .....	4194
Create .....	4196
BufferResize.....	4197
GetData .....	4198

Refresh .....	4200
CiTickVolume.....	4201
Create .....	4203
BufferResize.....	4204
GetData .....	4205
Refresh .....	4207
CiRealVolume.....	4208
Create .....	4210
BufferResize.....	4211
GetData .....	4212
Refresh .....	4214
CiOpen .....	4215
Create .....	4217
GetData .....	4218
CiHigh .....	4220
Create .....	4222
GetData .....	4223
CiLow .....	4225
Create .....	4227
GetData .....	4228
CiClose .....	4230
Create .....	4232
GetData .....	4233
Индикаторы тренда.....	4235
CiADX .....	4236
MaPeriod .....	4238
Create .....	4239
Main .....	4240
Plus .....	4241
Minus .....	4242
Type .....	4243
CiADXWilder.....	4244
MaPeriod .....	4246
Create .....	4247
Main .....	4248
Plus .....	4249
Minus .....	4250
Type .....	4251
CiBands .....	4252
MaPeriod .....	4254
MaShift .....	4255
Deviation .....	4256
Applied .....	4257
Create .....	4258
Base .....	4259
Upper .....	4260
Lower .....	4261
Type .....	4262
CiEnvelopes.....	4263
MaPeriod .....	4265
MaShift .....	4266
MaMethod .....	4267
Deviation .....	4268
Applied .....	4269
Create .....	4270
Upper .....	4271
Lower .....	4272
Type .....	4273

Cilchimoku.....	4274
TenkanSenPeriod.....	4276
KijunSenPeriod.....	4277
SenkouSpanBPeriod.....	4278
Create .....	4279
TenkanSen.....	4280
KijunSen .....	4281
SenkouSpanA.....	4282
SenkouSpanB.....	4283
ChinkouSpan.....	4284
Type .....	4285
CiMA .....	4286
MaPeriod .....	4288
MaShift .....	4289
MaMethod.....	4290
Applied .....	4291
Create .....	4292
Main .....	4293
Type .....	4294
CiSAR .....	4295
SarStep .....	4297
Maximum .....	4298
Create .....	4299
Main .....	4300
Type .....	4301
CiStdDev .....	4302
MaPeriod .....	4304
MaShift .....	4305
MaMethod.....	4306
Applied .....	4307
Create .....	4308
Main .....	4309
Type .....	4310
CiDEMA .....	4311
MaPeriod .....	4313
IndShift .....	4314
Applied .....	4315
Create .....	4316
Main .....	4317
Type .....	4318
CiTEMA .....	4319
MaPeriod .....	4321
IndShift .....	4322
Applied .....	4323
Create .....	4324
Main .....	4325
Type .....	4326
CiFrAMA .....	4327
MaPeriod .....	4329
IndShift .....	4330
Applied .....	4331
Create .....	4332
Main .....	4333
Type .....	4334
CiAMA .....	4335
MaPeriod .....	4337
FastEmaPeriod.....	4338
SlowEmaPeriod.....	4339

IndShift .....	4340
Applied .....	4341
Create .....	4342
Main .....	4343
Type .....	4344
CiVIDyA .....	4345
CmoPeriod.....	4347
EmaPeriod.....	4348
IndShift .....	4349
Applied .....	4350
Create .....	4351
Main .....	4352
Type .....	4353
Осцилляторы .....	4354
CiATR .....	4355
MaPeriod .....	4357
Create .....	4358
Main .....	4359
Type .....	4360
CiBearsPower .....	4361
MaPeriod .....	4363
Create .....	4364
Main .....	4365
Type .....	4366
CiBullsPower .....	4367
MaPeriod .....	4369
Create .....	4370
Main .....	4371
Type .....	4372
CiCCI .....	4373
MaPeriod .....	4375
Applied .....	4376
Create .....	4377
Main .....	4378
Type .....	4379
CiChaikin .....	4380
FastMaPeriod.....	4382
SlowMaPeriod.....	4383
MaMethod .....	4384
Applied .....	4385
Create .....	4386
Main .....	4387
Type .....	4388
CiDeMarker .....	4389
MaPeriod .....	4391
Create .....	4392
Main .....	4393
Type .....	4394
CiForce .....	4395
MaPeriod .....	4397
MaMethod .....	4398
Applied .....	4399
Create .....	4400
Main .....	4401
Type .....	4402
CiMACD .....	4403
FastEmaPeriod.....	4405
SlowEmaPeriod.....	4406

SignalPeriod.....	4407
Applied .....	4408
Create .....	4409
Main .....	4410
Signal .....	4411
Type .....	4412
<b>CiMomentum.....</b>	<b>4413</b>
MaPeriod .....	4415
Applied .....	4416
Create .....	4417
Main .....	4418
Type .....	4419
<b>CiOsMA .....</b>	<b>4420</b>
FastEmaPeriod.....	4422
SlowEmaPeriod.....	4423
SignalPeriod.....	4424
Applied .....	4425
Create .....	4426
Main .....	4427
Type .....	4428
<b>CiRSI .....</b>	<b>4429</b>
MaPeriod .....	4431
Applied .....	4432
Create .....	4433
Main .....	4434
Type .....	4435
<b>CiRVI .....</b>	<b>4436</b>
MaPeriod .....	4438
Create .....	4439
Main .....	4440
Signal .....	4441
Type .....	4442
<b>CiStochastic.....</b>	<b>4443</b>
Kperiod .....	4445
Dperiod .....	4446
Slowing .....	4447
MaMethod.....	4448
PriceField .....	4449
Create .....	4450
Main .....	4451
Signal .....	4452
Type .....	4453
<b>CiTriX .....</b>	<b>4454</b>
MaPeriod .....	4456
Applied .....	4457
Create .....	4458
Main .....	4459
Type .....	4460
<b>CiWPR .....</b>	<b>4461</b>
CalcPeriod.....	4463
Create .....	4464
Main .....	4465
Type .....	4466
<b>Индикаторы объема.....</b>	<b>4467</b>
<b>CiAD .....</b>	<b>4468</b>
Applied .....	4470
Create .....	4471
Main .....	4472

Type .....	4473
CiMFI .....	4474
MaPeriod .....	4476
Applied .....	4477
Create .....	4478
Main .....	4479
Type .....	4480
CiOBV .....	4481
Applied .....	4483
Create .....	4484
Main .....	4485
Type .....	4486
CiVolumes .....	4487
Applied .....	4489
Create .....	4490
Main .....	4491
Type .....	4492
Индикаторы Билла Вильямса .....	4493
CiAC .....	4494
Create .....	4496
Main .....	4497
Type .....	4498
CiAlligator .....	4499
JawPeriod .....	4501
JawShift .....	4502
TeethPeriod .....	4503
TeethShift .....	4504
LipsPeriod .....	4505
LipsShift .....	4506
MaMethod .....	4507
Applied .....	4508
Create .....	4509
Jaw .....	4510
Teeth .....	4511
Lips .....	4512
Type .....	4513
CiAO .....	4514
Create .....	4516
Main .....	4517
Type .....	4518
CiFractals .....	4519
Create .....	4521
Upper .....	4522
Lower .....	4523
Type .....	4524
CiGator .....	4525
JawPeriod .....	4527
JawShift .....	4528
TeethPeriod .....	4529
TeethShift .....	4530
LipsPeriod .....	4531
LipsShift .....	4532
MaMethod .....	4533
Applied .....	4534
Create .....	4535
Upper .....	4537
Lower .....	4538
Type .....	4539

CiBWMFI .....	4540
Applied .....	4542
Create .....	4543
Main .....	4544
Type .....	4545
Пользовательский индикатор.....	4546
NumBuffers.....	4547
NumParams.....	4548
ParamType.....	4549
ParamLong.....	4550
ParamDouble.....	4551
ParamString.....	4552
Type .....	4553
<b>Торговые классы .....</b>	<b>4554</b>
CAccountInfo.....	4555
Login .....	4558
TradeMode.....	4559
TradeModeDescription.....	4560
Leverage .....	4561
StopoutMode.....	4562
StopoutModeDescription.....	4563
MarginMode.....	4564
MarginModeDescription.....	4565
TradeAllowed.....	4566
TradeExpert.....	4567
LimitOrders.....	4568
Balance .....	4569
Credit .....	4570
Profit .....	4571
Equity .....	4572
Margin .....	4573
FreeMargin.....	4574
MarginLevel.....	4575
MarginCall.....	4576
MarginStopOut.....	4577
Name .....	4578
Server .....	4579
Currency .....	4580
Company .....	4581
InfoInteger.....	4582
InfoDouble.....	4583
InfoString .....	4584
OrderProfitCheck.....	4585
MarginCheck .....	4586
FreeMarginCheck .....	4587
MaxLotCheck .....	4588
CSymbolInfo.....	4589
Refresh .....	4594
RefreshRates.....	4595
Name .....	4596
Select .....	4597
IsSynchronized .....	4598
Volume .....	4599
VolumeHigh .....	4600
VolumeLow .....	4601
Time .....	4602
Spread .....	4603
SpreadFloat.....	4604

TicksBookDepth.....	4605
StopsLevel.....	4606
FreezeLevel.....	4607
Bid .....	4608
BidHigh .....	4609
BidLow .....	4610
Ask .....	4611
AskHigh .....	4612
AskLow .....	4613
Last .....	4614
LastHigh .....	4615
LastLow .....	4616
TradeCalcMode .....	4617
TradeCalcModeDescription.....	4618
TradeMode.....	4619
TradeModeDescription.....	4620
TradeExecution.....	4621
TradeExecutionDescription.....	4622
SwapMode .....	4623
SwapModeDescription.....	4624
SwapRollover3days .....	4625
SwapRollover3daysDescription.....	4626
MarginInitial.....	4627
MarginMaintenance.....	4628
MarginLong.....	4629
MarginShort .....	4630
MarginLimit.....	4631
MarginStop .....	4632
MarginStopLimit .....	4633
TradeTimeFlags .....	4634
TradeFillFlags.....	4635
Digits .....	4636
Point .....	4637
TickValue .....	4638
TickValueProfit.....	4639
TickValueLoss.....	4640
TickSize .....	4641
ContractSize.....	4642
LotsMin .....	4643
LotsMax .....	4644
LotsStep .....	4645
LotsLimit .....	4646
SwapLong .....	4647
SwapShort .....	4648
CurrencyBase.....	4649
CurrencyProfit.....	4650
CurrencyMargin.....	4651
Bank .....	4652
Description.....	4653
Path .....	4654
SessionDeals.....	4655
SessionBuyOrders.....	4656
SessionSellOrders.....	4657
SessionTurnover.....	4658
SessionInterest.....	4659
SessionBuyOrdersVolume.....	4660
SessionSellOrdersVolume .....	4661
SessionOpen.....	4662

SessionClose.....	4663
SessionAW .....	4664
SessionPriceSettlement.....	4665
SessionPriceLimitMin.....	4666
SessionPriceLimitMax.....	4667
InfoInteger.....	4668
InfoDouble.....	4669
InfoString .....	4670
NormalizePrice.....	4671
<b>COrderInfo.....</b>	<b>4672</b>
Ticket .....	4674
TimeSetup .....	4675
TimeSetupMsc.....	4676
OrderType.....	4677
TypeDescription.....	4678
State .....	4679
StateDescription.....	4680
TimeExpiration.....	4681
TimeDone .....	4682
TimeDoneMsc.....	4683
TypeFilling .....	4684
TypeFillingDescription.....	4685
TypeTime .....	4686
TypeTimeDescription.....	4687
Magic .....	4688
PositionId .....	4689
VolumeInitial.....	4690
VolumeCurrent.....	4691
PriceOpen.....	4692
StopLoss .....	4693
TakeProfit.....	4694
PriceCurrent.....	4695
PriceStopLimit .....	4696
Symbol .....	4697
Comment .....	4698
InfoInteger.....	4699
InfoDouble.....	4700
InfoString .....	4701
StoreState.....	4702
CheckState.....	4703
Select .....	4704
SelectByIndex.....	4705
<b>CHistoryOrderInfo.....</b>	<b>4706</b>
TimeSetup .....	4708
TimeSetupMsc.....	4709
OrderType.....	4710
TypeDescription.....	4711
State .....	4712
StateDescription.....	4713
TimeExpiration.....	4714
TimeDone .....	4715
TimeDoneMsc.....	4716
TypeFilling .....	4717
TypeFillingDescription.....	4718
TypeTime .....	4719
TypeTimeDescription.....	4720
Magic .....	4721
PositionId .....	4722

Volumelinitial.....	4723
VolumeCurrent.....	4724
PriceOpen.....	4725
StopLoss .....	4726
TakeProfit.....	4727
PriceCurrent.....	4728
PriceStopLimit.....	4729
Symbol .....	4730
Comment .....	4731
InfoInteger.....	4732
InfoDouble.....	4733
InfoString .....	4734
Ticket .....	4735
SelectByIndex.....	4736
<b>CPositionInfo.....</b>	<b>4737</b>
Time .....	4739
TimeMsc .....	4740
TimeUpdate.....	4741
TimeUpdateMsc.....	4742
PositionType .....	4743
TypeDescription.....	4744
Magic .....	4745
Identifier .....	4746
Volume .....	4747
PriceOpen.....	4748
StopLoss .....	4749
TakeProfit.....	4750
PriceCurrent.....	4751
Commission.....	4752
Swap .....	4753
Profit .....	4754
Symbol .....	4755
Comment .....	4756
InfoInteger.....	4757
InfoDouble.....	4758
InfoString .....	4759
Select .....	4760
SelectByIndex.....	4761
SelectByMagic.....	4762
SelectByTicket.....	4763
StoreState.....	4764
CheckState.....	4765
<b>CDealInfo.....</b>	<b>4766</b>
Order .....	4768
Time .....	4769
TimeMsc .....	4770
DealType .....	4771
TypeDescription.....	4772
Entry .....	4773
EntryDescription.....	4774
Magic .....	4775
PositionId .....	4776
Volume .....	4777
Price .....	4778
Commision .....	4779
Swap .....	4780
Profit .....	4781
Symbol .....	4782

Comment .....	4783
InfoInteger.....	4784
InfoDouble.....	4785
InfoString .....	4786
Ticket .....	4787
SelectByIndex.....	4788
<b>CTrade.....</b>	<b>4789</b>
LogLevel .....	4794
SetExpertMagicNumber .....	4795
SetDeviationInPoints.....	4796
SetTypeFilling .....	4797
SetTypeFillingBySymbol.....	4798
SetAsyncMode.....	4799
SetMarginMode.....	4800
OrderOpen.....	4801
OrderModify.....	4803
OrderDelete.....	4804
PositionOpen.....	4805
PositionModify .....	4806
PositionClose.....	4808
PositionClosePartial.....	4809
PositionCloseBy.....	4811
Buy .....	4812
Sell .....	4813
BuyLimit .....	4814
BuyStop .....	4815
SellLimit .....	4817
SellStop .....	4819
Request .....	4821
RequestAction .....	4822
RequestActionDescription.....	4823
RequestMagic .....	4824
RequestOrder .....	4825
RequestSymbol.....	4826
RequestVolume.....	4827
RequestPrice.....	4828
RequestStopLimit.....	4829
RequestSL .....	4830
RequestTP .....	4831
RequestDeviation .....	4832
RequestType .....	4833
RequestTypeDescription.....	4834
RequestTypeFilling .....	4835
RequestTypeFillingDescription.....	4836
RequestTypeTime .....	4837
RequestTypeTimeDescription.....	4838
RequestExpiration .....	4839
RequestComment.....	4840
RequestPosition.....	4841
RequestPositionBy .....	4842
Result .....	4843
ResultRetcode.....	4844
ResultRetcodeDescription.....	4845
ResultDeal .....	4846
ResultOrder .....	4847
ResultVolume .....	4848
ResultPrice .....	4849
ResultBid .....	4850

ResultAsk .....	4851
ResultComment .....	4852
CheckResult .....	4853
CheckResultRetcode.....	4854
CheckResultRetcodeDescription.....	4855
CheckResultBalance.....	4856
CheckResultEquity.....	4857
CheckResultProfit.....	4858
CheckResultMargin.....	4859
CheckResultMarginFree.....	4860
CheckResultMarginLevel.....	4861
CheckResultComment.....	4862
PrintRequest .....	4863
PrintResult.....	4864
FormatRequest .....	4865
FormatRequestResult.....	4866
CTerminalInfo.....	4867
Build .....	4870
IsConnected.....	4871
IsDLLsAllowed.....	4872
IsTradeAllowed.....	4873
IsEmailEnabled.....	4874
IsFtpEnabled.....	4875
MaxBars .....	4876
CodePage .....	4877
CPUcores .....	4878
MemoryPhysical.....	4879
MemoryTotal.....	4880
MemoryAvailable.....	4881
MemoryUsed.....	4882
IsX64 .....	4883
OpenCLSupport.....	4884
DiskSpace .....	4885
Language .....	4886
Name .....	4887
Company .....	4888
Path .....	4889
DataPath .....	4890
CommonDataPath.....	4891
InfoInteger.....	4892
InfoString .....	4893
<b>Модули стратегий .....</b>	<b>4894</b>
Базовые классы экспертов.....	4897
CExpertBase .....	4898
InitPhase .....	4901
TrendType.....	4902
UsedSeries.....	4903
EveryTick .....	4904
Open .....	4905
High .....	4906
Low .....	4907
Close .....	4908
Spread .....	4909
Time .....	4910
TickVolume.....	4911
RealVolume.....	4912
Init .....	4913
Symbol .....	4914

Period .....	4915
Magic .....	4916
ValidationSettings .....	4917
SetPriceSeries .....	4918
SetOtherSeries .....	4919
InitIndicators .....	4920
InitOpen .....	4921
InitHigh .....	4922
InitLow .....	4923
InitClose .....	4924
InitSpread .....	4925
InitTime .....	4926
InitTickVolume .....	4927
InitRealVolume .....	4928
PriceLevelUnit .....	4929
StartIndex .....	4930
CompareMagic .....	4931
<b>CExpert .....</b>	<b>4932</b>
Init .....	4937
Magic .....	4938
InitSignal .....	4939
InitTrailing .....	4940
InitMoney .....	4941
InitTrade .....	4942
Deinit .....	4943
OnTickProcess .....	4944
OnTradeProcess .....	4945
OnTimerProcess .....	4946
OnChartEventProcess .....	4947
OnBookEventProcess .....	4948
MaxOrders .....	4949
Signal .....	4950
ValidationSettings .....	4951
InitIndicators .....	4952
OnTick .....	4953
OnTrade .....	4954
OnTimer .....	4955
OnChartEvent .....	4956
OnBookEvent .....	4957
InitParameters .....	4958
DeinitTrade .....	4959
DeinitSignal .....	4960
DeinitTrailing .....	4961
DeinitMoney .....	4962
DeinitIndicators .....	4963
Refresh .....	4964
Processing .....	4965
SelectPosition .....	4967
CheckOpen .....	4968
CheckOpenLong .....	4969
CheckOpenShort .....	4970
OpenLong .....	4971
OpenShort .....	4972
CheckReverse .....	4973
CheckReverseLong .....	4974
CheckReverseShort .....	4975
ReverseLong .....	4976
ReverseShort .....	4978

CheckClose.....	4980
CheckCloseLong.....	4981
CheckCloseShort.....	4982
CloseAll .....	4983
Close .....	4984
CloseLong .....	4985
CloseShort.....	4986
CheckTrailingStop.....	4987
CheckTrailingStopLong.....	4988
CheckTrailingStopShort.....	4989
TrailingStopLong .....	4990
TrailingStopShort.....	4991
CheckTrailingOrderLong.....	4992
CheckTrailingOrderShort.....	4993
TrailingOrderLong .....	4994
TrailingOrderShort.....	4995
CheckDeleteOrderLong.....	4996
CheckDeleteOrderShort.....	4997
DeleteOrders .....	4998
DeleteOrder.....	4999
DeleteOrderLong.....	5000
DeleteOrderShort.....	5001
LotOpenLong .....	5002
LotOpenShort.....	5003
LotReverse.....	5004
PrepareHistoryDate.....	5005
HistoryPoint .....	5006
CheckTradeState.....	5007
WaitEvent.....	5008
NoWaitEvent.....	5009
TradeEventPositionStopTake.....	5010
TradeEventOrderTriggered.....	5011
TradeEventPositionOpened.....	5012
TradeEventPositionVolumeChanged.....	5013
TradeEventPositionModified.....	5014
TradeEventPositionClosed.....	5015
TradeEventOrderPlaced.....	5016
TradeEventOrderModified.....	5017
TradeEventOrderDeleted.....	5018
TradeEventNotIdentified.....	5019
TimeframeAdd.....	5020
TimeframesFlags.....	5021
CExpertSignal.....	5022
BasePrice .....	5025
UsedSeries.....	5026
Weight .....	5027
PatternsUsage.....	5028
General .....	5029
Ignore .....	5030
Invert .....	5031
ThresholdOpen.....	5032
ThresholdClose.....	5033
PriceLevel.....	5034
StopLevel .....	5035
TakeLevel .....	5036
Expiration.....	5037
Magic .....	5038
ValidationSettings .....	5039

InitIndicators.....	5040
AddFilter .....	5041
CheckOpenLong.....	5042
CheckOpenShort.....	5043
OpenLongParams.....	5044
OpenShortParams.....	5045
CheckCloseLong.....	5046
CheckCloseShort.....	5047
CloseLongParams.....	5048
CloseShortParams.....	5049
CheckReverseLong.....	5050
CheckReverseShort.....	5051
CheckTrailingOrderLong.....	5052
CheckTrailingOrderShort.....	5053
LongCondition.....	5054
ShortCondition.....	5055
Direction .....	5056
CExpertTrailing.....	5057
CheckTrailingStopLong.....	5059
CheckTrailingStopShort.....	5060
CExpertMoney .....	5061
Percent .....	5063
ValidationSettings .....	5064
CheckOpenLong.....	5065
CheckOpenShort.....	5066
CheckReverse.....	5067
CheckClose.....	5068
Модули торговых сигналов.....	5069
Сигналы индикатора Accelerator Oscillator.....	5072
Сигналы индикатора Adaptive Moving Average.....	5075
Сигналы индикатора Awesome Oscillator.....	5079
Сигналы осциллятора Bears Power.....	5083
Сигналы осциллятора Bulls Power.....	5085
Сигналы осциллятора Commodity Channel Index.....	5087
Сигналы осциллятора DeMarker.....	5091
Сигналы индикатора Double Exponential Moving Average.....	5095
Сигналы индикатора Envelopes .....	5099
Сигналы индикатора Fractal Adaptive Moving Average.....	5102
Сигналы внутридневного временного фильтра.....	5106
Сигналы осциллятора MACD .....	5108
Сигналы индикатора Moving Average.....	5114
Сигналы индикатора Parabolic SAR.....	5118
Сигналы осциллятора Relative Strength Index.....	5120
Сигналы осциллятора Relative Vigor Index.....	5126
Сигналы осциллятора Stochastic.....	5128
Сигналы осциллятора Triple Exponential Average.....	5133
Сигналы индикатора Triple Exponential Moving Average.....	5137
Сигналы осциллятора Williams Percent Range.....	5141
Модули Trailing Stop.....	5144
CTrailingFixedPips.....	5145
StopLevel .....	5147
ProfitLevel.....	5148
ValidationSettings .....	5149
CheckTrailingStopLong.....	5150
CheckTrailingStopShort.....	5151
CTrailingMA.....	5152
Period .....	5154
Shift .....	5155

Method .....	5156
Applied .....	5157
InitIndicators.....	5158
ValidationSettings .....	5159
CheckTrailingStopLong.....	5160
CheckTrailingStopShort.....	5161
CTrailingNone.....	5162
CheckTrailingStopLong.....	5163
CheckTrailingStopShort.....	5164
CTrailingPSAR.....	5165
Step .....	5167
Maximum .....	5168
InitIndicators.....	5169
CheckTrailingStopLong.....	5170
CheckTrailingStopShort.....	5171
Модули Money Management.....	5172
CMoneyFixedLot.....	5173
Lots .....	5175
ValidationSettings .....	5176
CheckOpenLong.....	5177
CheckOpenShort .....	5178
CMoneyFixedMargin.....	5179
CheckOpenLong.....	5180
CheckOpenShort .....	5181
CMoneyFixedRisk.....	5182
CheckOpenLong.....	5183
CheckOpenShort .....	5184
CMoneyNone .....	5185
ValidationSettings .....	5186
CheckOpenLong.....	5187
CheckOpenShort .....	5188
CMoneySizeOptimized.....	5189
DecreaseFactor.....	5191
ValidationSettings .....	5192
CheckOpenLong.....	5193
CheckOpenShort .....	5194
Панели и диалоги .....	5195
CRect .....	5197
Left .....	5198
Top .....	5199
Right .....	5200
Bottom .....	5201
Width .....	5202
Height .....	5203
SetBound .....	5204
Move .....	5205
Shift .....	5206
Contains .....	5207
Format .....	5208
CDateTime.....	5209
MonthName.....	5211
ShortMonthName.....	5212
DayName .....	5213
ShortDayName.....	5214
DaysInMonth.....	5215
DateTime .....	5216
Date .....	5217
Time .....	5218

Sec .....	5219
Min .....	5220
Hour .....	5221
Day .....	5222
Mon .....	5223
Year .....	5224
SecDec .....	5225
SecInc .....	5226
MinDec .....	5227
MinInc .....	5228
HourDec .....	5229
HourInc .....	5230
DayDec .....	5231
DayInc .....	5232
MonDec .....	5233
MonInc .....	5234
YearDec .....	5235
YearInc .....	5236
<b>CWnd .....</b>	<b>5237</b>
Create .....	5241
Destroy .....	5242
OnEvent .....	5243
OnMouseEvent .....	5244
Name .....	5245
ControlsTotal .....	5246
Control .....	5247
ControlFind .....	5248
Rect .....	5249
Left .....	5250
Top .....	5251
Right .....	5252
Bottom .....	5253
Width .....	5254
Height .....	5255
Move .....	5256
Shift .....	5257
Resize .....	5258
Contains .....	5259
Alignment .....	5260
Align .....	5261
Id .....	5262
IsEnabled .....	5263
Enable .....	5264
Disable .....	5265
IsVisible .....	5266
Visible .....	5267
Show .....	5268
Hide .....	5269
IsActive .....	5270
Activate .....	5271
Deactivate .....	5272
StateFlags .....	5273
StateFlagsSet .....	5274
StateFlagsReset .....	5275
PropFlags .....	5276
PropFlagsSet .....	5277
PropFlagsReset .....	5278
MouseX .....	5279

MouseY .....	5280
MouseFlags .....	5281
MouseFocusKill.....	5282
OnCreate .....	5283
OnDestroy.....	5284
OnMove .....	5285
OnResize .....	5286
OnEnable .....	5287
OnDisable .....	5288
OnShow .....	5289
OnHide .....	5290
OnActivate.....	5291
OnDeactivate.....	5292
OnClick .....	5293
OnChange .....	5294
OnMouseDown.....	5295
OnMouseUp.....	5296
OnDragStart.....	5297
OnDragProcess.....	5298
OnDragEnd.....	5299
DragObjectCreate.....	5300
DragObjectDestroy.....	5301
CWndObj .....	5302
OnEvent .....	5305
Text .....	5306
Color .....	5307
ColorBackground.....	5308
ColorBorder.....	5309
Font .....	5310
FontSize .....	5311
ZOrder .....	5312
OnObjectCreate.....	5313
OnObjectChange.....	5314
OnObjectDelete.....	5315
OnObjectDrag.....	5316
OnSetText .....	5317
OnSetColor .....	5318
OnSetColorBackground.....	5319
OnSetFont .....	5320
OnSetFontSize.....	5321
OnSetZOrder.....	5322
OnDestroy.....	5323
OnChange .....	5324
CWndContainer .....	5325
Destroy .....	5328
OnEvent .....	5329
OnMouseEvent.....	5330
ControlsTotal.....	5331
Control .....	5332
ControlFind.....	5333
Add .....	5334
Delete .....	5335
Move .....	5336
Shift .....	5337
Id .....	5338
Enable .....	5339
Disable .....	5340
Show .....	5341

Hide .....	5342
MouseFocusKill.....	5343
Save .....	5344
Load .....	5345
OnResize .....	5346
OnActivate.....	5347
OnDeactivate.....	5348
CLabel .....	5349
Create .....	5354
OnSetText.....	5355
OnSetColor.....	5356
OnSetFont.....	5357
OnSetFontSize.....	5358
OnCreate .....	5359
OnShow .....	5360
OnHide .....	5361
OnMove .....	5362
CBmpButton.....	5363
Create .....	5370
Border .....	5371
BmpNames .....	5372
BmpOffName.....	5373
BmpOnName.....	5374
BmpPassiveName.....	5375
BmpActiveName.....	5376
Pressed .....	5377
Locking .....	5378
OnSetZOrder.....	5379
OnCreate .....	5380
OnShow .....	5381
OnHide .....	5382
OnMove .....	5383
OnChange .....	5384
OnActivate.....	5385
OnDeactivate.....	5386
OnMouseDown.....	5387
OnMouseUp.....	5388
CButton.....	5389
Create .....	5396
Pressed .....	5397
Locking .....	5398
OnSetText.....	5399
OnSetColor.....	5400
OnSetColorBackground.....	5401
OnSetColorBorder.....	5402
OnSetFont .....	5403
OnSetFontFontSize.....	5404
OnCreate .....	5405
OnShow .....	5406
OnHide .....	5407
OnMove .....	5408
OnResize .....	5409
OnMouseDown.....	5410
OnMouseUp.....	5411
CEdit .....	5412
Create .....	5418
ReadOnly .....	5419
TextAlign .....	5420

OnObjectEndEdit.....	5421
OnSetText.....	5422
OnSetColor.....	5423
OnSetColorBackground.....	5424
OnSetColorBorder.....	5425
OnSetFont.....	5426
OnFontSize.....	5427
OnSetZOrder.....	5428
OnCreate .....	5429
OnShow .....	5430
OnHide .....	5431
OnMove .....	5432
OnResize .....	5433
OnChange .....	5434
OnClick .....	5435
<b>CPanel .....</b>	<b>5436</b>
Create .....	5441
BorderType.....	5442
OnSetText.....	5443
OnSetColorBackground.....	5444
OnSetColorBorder.....	5445
OnCreate .....	5446
OnShow .....	5447
OnHide .....	5448
OnMove .....	5449
OnResize .....	5450
OnChange .....	5451
<b>CPicture.....</b>	<b>5452</b>
Create .....	5457
Border .....	5458
BmpName .....	5459
OnCreate .....	5460
OnShow .....	5461
OnHide .....	5462
OnMove .....	5463
OnChange .....	5464
<b>CScroll .....</b>	<b>5465</b>
Create .....	5468
OnEvent .....	5469
MinPos .....	5470
MaxPos .....	5471
CurrPos .....	5472
CreateBack.....	5473
CreateInc .....	5474
CreateDec.....	5475
CreateThumb.....	5476
OnClickInc.....	5477
OnClickDec .....	5478
OnShow .....	5479
OnHide .....	5480
OnChangePos .....	5481
OnThumbDragStart .....	5482
OnThumbDragProcess .....	5483
OnThumbDragEnd .....	5484
CalcPos .....	5485
<b>CScrollView .....</b>	<b>5486</b>
CreateInc .....	5492
CreateDec .....	5493

CreateThumb.....	5494
OnResize .....	5495
OnChangePos.....	5496
OnThumbDragStart.....	5497
OnThumbDragProcess.....	5498
OnThumbDragEnd.....	5499
CalcPos .....	5500
CScrollH.....	5501
CreateInc .....	5507
CreateDec.....	5508
CreateThumb.....	5509
OnResize .....	5510
OnChangePos.....	5511
OnThumbDragStart.....	5512
OnThumbDragProcess.....	5513
OnThumbDragEnd.....	5514
CalcPos .....	5515
CWndClient.....	5516
Create .....	5519
OnEvent .....	5520
ColorBackground.....	5521
ColorBorder.....	5522
BorderType.....	5523
VScrolled .....	5524
HScrolled .....	5525
CreateBack.....	5526
CreateScrollV.....	5527
CreateScrollH.....	5528
OnResize .....	5529
OnVScrollShow.....	5530
OnVScrollHide.....	5531
OnHScrollShow.....	5532
OnHScrollHide.....	5533
OnScrollLineDown.....	5534
OnScrollLineUp.....	5535
OnScrollLineLeft.....	5536
OnScrollLineRight.....	5537
Rebound .....	5538
CLListView.....	5539
Create .....	5545
OnEvent .....	5546
TotalView .....	5547
AddItem .....	5548
Select .....	5549
SelectByText .....	5550
SelectByValue .....	5551
Value .....	5552
CreateRow.....	5553
OnResize .....	5554
OnVScrollShow.....	5555
OnVScrollHide.....	5556
OnScrollLineDown.....	5557
OnScrollLineUp.....	5558
OnItemClick.....	5559
Redraw .....	5560
RowState .....	5561
CheckView.....	5562
CComboBox.....	5563

Create .....	5569
OnEvent .....	5570
AddItem .....	5571
ListViewItems.....	5572
Select .....	5573
SelectByText.....	5574
SelectByValue.....	5575
Value .....	5576
CreateEdit.....	5577
CreateButton.....	5578
CreateList.....	5579
OnClickEdit.....	5580
OnClickButton.....	5581
OnChangeList.....	5582
ListShow .....	5583
ListHide .....	5584
<b>CCheckBox.....</b>	<b>5585</b>
Create .....	5592
OnEvent .....	5593
Text .....	5594
Color .....	5595
Checked .....	5596
Value .....	5597
CreateButton.....	5598
CreateLabel.....	5599
OnClickButton.....	5600
OnClickLabel.....	5601
<b>CCheckGroup.....</b>	<b>5602</b>
Create .....	5608
OnEvent .....	5609
AddItem .....	5610
Value .....	5611
CreateButton.....	5612
OnVScrollShow.....	5613
OnVScrollHide.....	5614
OnScrollLineDown.....	5615
OnScrollLineUp.....	5616
OnChangeItem.....	5617
Redraw .....	5618
RowState .....	5619
<b>CRadioButton.....</b>	<b>5620</b>
Create .....	5622
OnEvent .....	5623
Text .....	5624
Color .....	5625
State .....	5626
CreateButton.....	5627
CreateLabel.....	5628
OnClickButton.....	5629
OnClickLabel.....	5630
<b>CRadioGroup.....</b>	<b>5631</b>
Create .....	5637
OnEvent .....	5638
AddItem .....	5639
Value .....	5640
CreateButton.....	5641
OnVScrollShow.....	5642
OnVScrollHide.....	5643

OnScrollLineDown.....	5644
OnScrollLineUp.....	5645
OnChangeItem.....	5646
Redraw .....	5647
RowState .....	5648
Select .....	5649
<b>CSpinEdit.....</b>	<b>5650</b>
Create .....	5656
OnEvent .....	5657
MinValue .....	5658
MaxValue .....	5659
Value .....	5660
CreateEdit.....	5661
CreateInc .....	5662
CreateDec.....	5663
OnClickInc.....	5664
OnClickDec .....	5665
OnChangeValue.....	5666
<b>CDialog.....</b>	<b>5667</b>
Create .....	5670
OnEvent .....	5671
Caption .....	5672
Add .....	5673
CreateWhiteBorder.....	5674
CreateBackground.....	5675
CreateCaption.....	5676
CreateButtonClose.....	5677
CreateClientArea.....	5678
OnClickCaption.....	5679
OnClickButtonClose.....	5680
ClientAreaVisible .....	5681
ClientAreaLeft .....	5682
ClientAreaTop .....	5683
ClientAreaRight .....	5684
ClientAreaBottom .....	5685
ClientAreaWidth .....	5686
ClientAreaHeight .....	5687
OnDialogDragStart .....	5688
OnDialogDragProcess .....	5689
OnDialogDragEnd .....	5690
<b>CAppDialog .....</b>	<b>5691</b>
Create .....	5694
Destroy .....	5695
OnEvent .....	5696
Run .....	5697
ChartEvent .....	5698
Minimized .....	5699
IniFileSave .....	5700
IniFileLoad .....	5701
IniFileName .....	5702
IniFileExt .....	5703
CreateCommon .....	5704
CreateExpert .....	5705
CreateIndicator .....	5706
CreateButtonMinMax .....	5707
OnClickButtonClose .....	5708
OnClickButtonMinMax .....	5709
OnAnotherApplicationClose .....	5710

Rebound .....	5711
Minimize .....	5712
Maximize .....	5713
CreateInstanceld.....	5714
ProgramName.....	5715
SubwinOff .....	5716
<b>32 Переход с MQL4.....</b>	<b>5717</b>
<b>33 Список функций языка MQL5.....</b>	<b>5721</b>
<b>34 Список констант языка MQL5.....</b>	<b>5756</b>

## Справочник MQL5

MetaQuotes Language 5 (MQL5) - язык программирования технических индикаторов, торговых роботов и вспомогательных приложений для автоматизации торговли на финансовых рынках. MQL5 является современным языком высокого уровня и разработан [MetaQuotes Software Corp.](#) для собственной торгово-информационной платформы. Синтаксис языка максимально близок к C++ и позволяет писать программы в стиле объектно-ориентированного программирования (ООП).

Для написания программ на MQL5 в составе торговой платформы предоставляется [среда разработки MetaEditor](#) со всеми современными инструментами для написания кода, включающими в себя шаблоны, сниппеты, отладку, профилировку, автозавершение и встроенное версионное хранилище [MQL5 Storage](#).

Поддержка и развитие языка осуществляется на сайте MQL5.community, где находится обширная [библиотека бесплатных кодов](#) и множество [статей](#). Эти статьи охватывают все темы современного трейдинга: нейронные сети, статистика и анализ, высокочастотная торговля, арбитраж, тестирование и оптимизация торговых стратегий, использование роботов для автоматизации торговли и многое другое.

Трейдеры и разработчики MQL5-программ могут общаться на форуме, проводить заказы во [Фрилансе](#), покупать и продавать защищенные программы в [Маркете](#) - магазине готовых приложений для автотрейдинга.

Язык MQL5 содержит специализированные [торговые функции](#) и предопределенные [обработчики событий](#) для написания советников (Expert Advisors). Советники автоматически управляют торговыми процессами на основе заложенных в них торговых правил. Также на MQL5 можно создавать собственные [технические индикаторы](#) (Custom Indicators), скрипты (Scripts) и библиотеки функций (Libraries).

Справочник MQL5 содержит разбитые на категории функции, операции, зарезервированные слова, другие конструкции языка и позволяет узнать описание каждого используемого элемента, входящего в состав языка. Также в справочнике приведено описание классов из состава [Стандартной библиотеки](#) для создания торговых стратегий, панелей управления, пользовательской графики и работы с файлами.

Отдельно от справочника в CodeBase опубликована библиотека численного анализа [ALGLIB](#), которая позволяет решать множество математических задач.

## Виды приложений в MQL5

Для выполнения конкретных задач по автоматизации торговых операций MQL5-программы разделены на четыре специализированных типа:

- **Советник** – автоматическая торговая система, имеющая привязку к определенному графику. Советник содержит в себе функции-обработчики предопределенных [событий](#), при наступлении которых выполняются соответствующие элементы торговой стратегии. Примеры таких событий – инициализация и деинициализация программы, приход нового тика, срабатывание таймера, изменение в стакане цен, события графика и пользовательские события. Советник может не только вычислять торговые сигналы по заложенным правилам, но и автоматически совершать сделки на торговом счете, направляя их прямо на торговый сервер. Советники хранятся в директории <каталог\_терминала>\MQL5\Experts.

- **Пользовательский индикатор** — это технический индикатор, написанный пользователем в дополнение к индикаторам, уже интегрированным в торговую платформу. Пользовательские индикаторы, также как и встроенные, не могут автоматически торговать и предназначены только для реализации аналитических функций. Пользовательские индикаторы могут использовать в своих расчетах значения других индикаторов, а также сами могут вызываться в советниках.

Пользовательские индикаторы хранятся в директории <каталог\_терминала>\MQL5\Indicators.

- **Скрипт** — программа, предназначенная для одноразового выполнения каких-либо действий. В отличие от экспертов, скрипты не обрабатывают никаких событий, кроме события запуска. Для работы скрипта в его коде обязательно должна быть функция-обработчик OnStart. Скрипты хранятся в директории <каталог\_терминала>\MQL5\Scripts.
- **Сервис** — программа, которая в отличие от индикаторов, советников и скриптов для своей работы не требует привязки к графику. Как и скрипты, сервисы не обрабатывают никаких событий, кроме события запуска. Для запуска сервиса в его коде обязательно должна быть функция-обработчик OnStart. Сервисы не принимают никаких других событий кроме Start, но могут сами отправлять графикам пользовательские события с помощью [EventChartCustom](#). Сервисы хранятся в директории <каталог\_терминала>\MQL5\Services.
- **Библиотека** — библиотека пользовательских функций, предназначенная для хранения и распространения часто используемых блоков пользовательских программ. Библиотеки не могут самостоятельно запускаться на выполнение.  
Библиотеки хранятся в директории <каталог\_терминала>\MQL5\Libraries
- **Включаемый файл** — 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 могут находиться в той же директории, что и исходный файл, в этом случае используется директива `#include` с двойными кавычками. Другое место хранения включаемых файлов - в директории <каталог\_терминала>\MQL5\Include, в этом случае используется директива `#include` с угловыми скобками.

© 2000-2019, [MetaQuotes Software Corp.](#)

## Основы языка

Язык MetaQuotes Language 5 (MQL5) является объектно-ориентированным языком программирования высокого уровня и предназначен для написания автоматических торговых стратегий, пользовательских технических индикаторов для анализа разнообразных финансовых рынков. Он позволяет не только писать разнообразные экспертные системы, предназначенные для работы в режиме реального времени, но и создавать собственные графические инструменты, помогающие принимать торговые решения.

MQL5 основан на концепции широко распространенного языка программирования C++, по сравнению с MQL4 в нем добавлены [перечисления](#), [структуры](#), [классы](#) и [обработка событий](#). Благодаря расширению числа встроенных основных [типов](#), взаимодействие исполняемых программ на MQL5 с другими приложениями посредством dll максимально облегчено. Синтаксис языка MQL5 подобен синтаксису C++, и это позволяет легко переносить на него программы из современных языков программирования.

Для целей изучения языка MQL5 все темы сгруппированы по следующим разделам:

- [Синтаксис](#)
- [Типы данных](#)
- [Операции и выражения](#)
- [Операторы](#)
- [Функции](#)
- [Переменные](#)
- [Препроцессор](#)
- [Объектно-ориентированное программирование](#)

## Синтаксис

Синтаксически язык программирования торговых стратегий MQL5 очень похож на язык программирования C++, за исключением некоторых возможностей:

- отсутствует адресная арифметика;
- отсутствует оператор goto;
- нельзя объявить анонимное перечисление;
- нет множественного наследования.

Смотри также

[Перечисления](#), [Структуры и классы](#), [Наследование](#)

## 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
 * Многостроч-
     ный           // Вложенный однострочный комментарий
     комментарий
 */
```

## Идентификаторы

Идентификаторы используются в качестве имен для переменных и функций. Длина идентификатора не может превышать 63 знака.

Допустимые символы при написании идентификатора: 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 namel Total_5 Paper
```

**Смотри также**

[Переменные](#), [Функции](#)

## Зарезервированные слова

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

### Типы данных

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

### Спецификаторы доступа

<u>const</u>	<u>private</u>	<u>protected</u>
<u>public</u>	<u>virtual</u>	

### Классы памяти

<u>extern</u>	<u>input</u>	<u>static</u>
---------------	--------------	---------------

### Операторы

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

### Прочие

<u>false</u>	<u>#define</u>	<u>#property</u>
<u>this</u>	<u>#import</u>	<u>template</u>
<u>true</u>	<u>#include</u>	<u>typename</u>

## Типы данных

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 MQL5 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 ([char](#), [short](#), [int](#), [long](#), [uchar](#), [ushort](#), [uint](#), [ulong](#))
- логические ([bool](#))
- [литералы](#) ([ushort](#))
- строки ([string](#))
- с плавающей точкой ([double](#), [float](#))
- цвет ([color](#))
- дата и время ([datetime](#))
- перечисления ([enum](#))

Сложные типы данных:

- [структуры](#);
- [классы](#).

В терминах [ООП](#) сложные типы данных называются абстрактными типами данных.

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 неявное [приведение типов](#), если не указано явное приведение.

Смотри также

[Приведение типов](#)

## Целые типы

Целые типы представлены в языке MQL5 одиннадцатью видами. Некоторые из типов могут использоваться вместе с другими, если этого требует логика программы, но при этом необходимо иметь ввиду правила [преобразования типов](#).

В таблице приведены характеристики каждого типа. Кроме того, в последнем столбце для каждого типа указан соответствующий тип в языке программирования C++.

Тип	Размер в байтах	Минимальное значение	Максимальное значение	Аналог в языке C++
<a href="#">char</a>	1	-128	127	char
<a href="#">uchar</a>	1	0	255	unsigned char, BYTE
<a href="#">bool</a>	1	0(false)	1(true)	bool
<a href="#">short</a>	2	-32 768	32 767	short, wchar_t
<a href="#">ushort</a>	2	0	65 535	unsigned short, WORD
<a href="#">int</a>	4	- 2 147 483 648	2 147 483 647	int
<a href="#">uint</a>	4	0	4 294 967 295	unsigned int, DWORD
<a href="#">color</a>	4	-1	16 777 215	int, COLORREF
<a href="#">long</a>	8	-9 223 372 036 854 775 808	9 223 372 036 854 775 807	<a href="#">__int64</a>
<a href="#">ulong</a>	8	0	18 446 744 073 709 551 615	unsigned <a href="#">__int64</a>
<a href="#">datetime</a>	8	0 (1970.01.01 0:00:00)	32 535 244 799 (3000.12.31 23:59:59)	<a href="#">__time64_t</a>

Значения целых типов можно также представлять в виде числовых констант, цветовых литералов, литералов даты-времени, [символьных констант](#) и [перечислений](#).

### Смотри также

[Преобразование данных](#), [Константы числовых типов](#)

## Типы `char`, `short`, `int` и `long`

### `char`

Целый тип `char` занимает в памяти 1 байт (8 бит) и позволяет выразить в двоичной системе счисления  $2^8$  значений=256. Тип `char` может содержать как положительные, так и отрицательные значения. Диапазон изменения значений составляет от -128 до 127.

### `uchar`

Целый тип `uchar` также занимает в памяти 1 байт, как и тип `char`, но в отличие от него, `uchar` предназначен только для положительных значений. Минимальное значение равно нулю, максимальное значение равно 255. Первая буква `u` в названии типа `uchar` является сокращением слова `unsigned` (беззнаковый).

### `short`

Целый тип `short` имеет размер 2 байта(16 бит) и, соответственно, позволяет выразить множество значений равное 2 в степени 16:  $2^{16}=65\,536$ . Так как тип `short` является знаковым и содержит как положительные, так и отрицательные значения, то диапазон значений находится между -32 768 и 32 767.

### `ushort`

Беззнаковым типом `short` является тип `ushort`, который также имеет размер 2 байта. Минимальное значение равно 0, максимальное значение 65 535.

### `int`

Целый тип `int` имеет размер 4 байта (32 бита). Минимальное значение -2 147 483 648, максимальное значение 2 147 483 647.

### `uint`

Беззнаковый целый тип `uint` занимает в памяти 4 байта и позволяет выражать целочисленные значения от 0 до 4 294 967 295.

### `long`

Целый тип `long` имеет размер 8 байт (64 бита). Минимальное значение -9 223 372 036 854 775 808, максимальное значение 9 223 372 036 854 775 807.

### `ulong`

Целый тип `ulong` также занимает 8 байт и позволяет хранить значения от 0 до 18 446 744 073 709 551 615.

Примеры:

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

Так как беззнаковые целые типы не предназначены для хранения отрицательных значений, то попытка установить отрицательное значение может привести к неожиданным последствиям. Вот такой невинный скрипт приведет к бесконечному циклу:

```
//--- бесконечный цикл
void OnStart()
{
    uchar u_ch;

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

Правильно будет так:

```
//--- правильный вариант
void OnStart()
{
    uchar u_ch;

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

**Результат:**

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

```

ch= -113  u_ch= 143
ch= -112  u_ch= 144
ch= -111  u_ch= 145
...

```

**Примеры:**

```

//--- отрицательные значения нельзя хранить в беззнаковых типах
uchar  u_ch=-120;
ushort u_sh=-5000;
uint   u_in=-401280;

```

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
```

Для целочисленных переменных значения можно задавать в бинарном виде с помощью префикса B. Например, можно закодировать рабочие часы торговой сессии в переменную типа **int** и использовать информацию о них согласно требуемому алгоритму:

```

//-----+
//| Script program start function
//-----+
void OnStart()
{
//--- для рабочих часов ставим 1, для нерабочих указываем 0
    int AsianSession =B'111111111'; // азиатская сессия с 0:00 часов до 9:00
    int EuropeanSession=B'11111111000000000'; // европейская сессия 9:00 - 18:00
    int AmericanSession =B'111111110000000000000001'; // американская 16:00 - 02:00
//--- выведем числовые значения сессий
    PrintFormat("Asian session hours as value =%d",AsianSession);
    PrintFormat("European session hours as value is %d",EuropeanSession);
    PrintFormat("American session hours as value is %d",AmericanSession);
//--- а теперь выведем строковые представления рабочих часов сессий
    Print("Asian session ",GetHoursForSession(AsianSession));
    Print("European session ",GetHoursForSession(EuropeanSession));
    Print("American session ",GetHoursForSession(AmericanSession));
//---
}
//-----+
//| возвращает рабочие часы сессии в строковом виде
//-----+
string GetHoursForSession(int session)
{
//--- для проверки используем битовые операции AND и сдвиг на один бит влево <<=1

```

```
//--- начинаем проверять с самого младшего бита
int bit=1;
string out="working hours: ";
//--- будем проверять все 24 бита, начиная с нулевого до 23 включительно
for(int i=0;i<24;i++)
{
    //--- получим состояние бита bit в числе number
    bool workinghour=(session&bit)==bit;
    //--- добавим номер часа в сообщение
    if(workinghour)out=out+StringFormat("%d ",i);
    //--- сдвигаем на один бит влево для проверки значения следующего
    bit<<=1;
}
//--- сформированная строка
return out;
}
```

#### Смотри также

[Приведение типов](#)

## Символьные константы

Символы, как элемент [строки](#), в MQL5 - это индексы в наборе символов Unicode. Они являются 16-разрядными значениями, которые можно преобразовывать в целые числа и с которыми можно манипулировать целочисленными [операциями](#), такими как сложение и вычитание.

Любой одиночный символ, заключенный в одинарные кавычки, или шестнадцатеричный ASCII-код символа в виде '\x10' является символьной константой и имеет тип [ushort](#). Например, запись вида '0' представляет из себя числовое значение 30, соответствующее индексу, по которому в таблице символов располагается символ ноль.

**Пример:**

```
void OnStart()
{
    //--- определим символьные константы
    int symbol_0='0';
    int symbol_9=symbol_0+9; // получим символ '9'
    //--- выведем значения констант
    printf("В десятичном виде: symbol_0 = %d, symbol_9 = %d",symbol_0,symbol_9);
    printf("В шестнадцатеричном виде: symbol_0 = 0x%x, symbol_9 = 0x%x",symbol_0,symbol_9);
    //--- заведем константы в строку
    string test="";
    StringSetCharacter(test,0,symbol_0);
    StringSetCharacter(test,1,symbol_9);
    //--- а вот как они выглядят в строке
    Print(test);
}
```

Обратная косая черта является управляющим символом для компилятора при разборе константных строк и символьных констант в исходном тексте программы. Некоторые символы, например, одинарные кавычки ('), двойные кавычки ("), обратная косая черта (\) и управляющие символы можно представлять комбинацией символов, начинающейся с обратной косой черты(\), в соответствии с приводимой ниже таблицей:

Название символа	Мнемокод или изображение	Запись в MQL5	Числовое значение
новая строка (перевод строки)	LF	'\n'	10
горизонтальная табуляция	HT	'\t'	9
возврат каретки	CR	'\r'	13
обратная косая черта	\	'\\'	92
одинарная кавычка	'	'\"''	39
двойная кавычка	"	'\"'''	34

шестнадцатеричный код	hhhh	'\xhhhh'	от 1 до 4 шестнадцатеричных знаков
десятичный код	d	'\d'	десятичное число от 0 до 65535

Если за обратной косой чертой следует символ, отличный от перечисленных, результат не определен.

### Пример

```
void OnStart()
{
//--- объявим символьные константы
int a='A';
int b='$';
int c='®'; // код 0xA9
int d='\xAE'; // код символа ®
//--- выведем константы на печать
Print(a,b,c,d);
//--- добавим символ в строку
string test="";
StringSetCharacter(test,0,a);
Print(test);
//--- заменим символ в строке
StringSetCharacter(test,0,b);
Print(test);
//--- заменим символ в строке
StringSetCharacter(test,0,c);
Print(test);
//--- заменим символ в строке
StringSetCharacter(test,0,d);
Print(test);
//--- представим символы в виде числа
int a1=65;
int b1=36;
int c1=169;
int d1=174;
//--- добавим символ в строку
StringSetCharacter(test,1,a1);
Print(test);
//--- заменим символ в строке
StringSetCharacter(test,1,b1);
Print(test);
//--- заменим символ в строке
StringSetCharacter(test,1,c1);
Print(test);
//--- заменим символ в строке
```

```

StringSetCharacter(test,1,d1);
Print(test);
}

```

Как уже говорилось выше, значение символьной константы (или переменной) представляет собой индекс в таблице символов, а так как индекс является целым числом, то допустимо его записывать разными способами.

```

void OnStart()
{
//---
int a=0xAE;           // код символа ® соответствует литералу '\xAE'
int b=0x24;           // код символа $ соответствует литералу '\x24'
int c=0xA9;           // код символа © соответствует литералу '\xA9'
int d=0x263A;         // код символа ™ соответствует литералу '\x263A'

//--- выведем значения
Print(a,b,c,d);

//--- добавим символ в строку
string test="";
StringSetCharacter(test,0,a);
Print(test);

//--- заменим символ в строке
StringSetCharacter(test,0,b);
Print(test);

//--- заменим символ в строке
StringSetCharacter(test,0,c);
Print(test);

//--- заменим символ в строке
StringSetCharacter(test,0,d);
Print(test);

//--- коды мастей
int a1=0x2660;
int b1=0x2661;
int c1=0x2662;
int d1=0x2663;

//--- добавим символ пикей
StringSetCharacter(test,1,a1);
Print(test);

//--- добавим символ червей
StringSetCharacter(test,2,b1);
Print(test);

//--- добавим символ бубей
StringSetCharacter(test,3,c1);
Print(test);

//--- добавим символ треф
StringSetCharacter(test,4,d1);
Print(test);

//--- Пример символьных литералов в строке
test="Дама\x2660Туз\x2662";
}

```

```
    printf("%s",test);
}
```

Внутреннее представление символьного литерала - тип [ushort](#). Символьные константы могут принимать значения от 0 до 65535.

#### Смотри также

[StringSetCharacter\(\)](#), [StringGetCharacter\(\)](#), [ShortToString\(\)](#), [ShortArrayToString\(\)](#),  
[StringToShortArray\(\)](#)

## Тип `datetime`

Тип `datetime` предназначен для хранения даты и времени в виде количества секунд, прошедших с 01 января 1970 года. Занимает в памяти 8 байт.

Константы даты и времени могут быть представлены в виде литературной 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.

Диапазон значений от 1 января 1970 года до 31 декабря 3000 года. Может опускаться либо дата (год, месяц, число), либо время (часы, минуты, секунды), либо все вместе.

При литературальном задании даты желательно указывать год, месяц и день, иначе компилятор выдаст [предупреждения](#) о неполной литературальной записи.

**Примеры:**

```
datetime NY=D'2015.01.01 00:00';           // время наступления 2015 года
datetime d1=D'1980.07.19 12:30:27';       // год месяц день часы минуты секунды
datetime d2=D'19.07.1980 12:30:27';        // равнозначно D'1980.07.19 12:30:27';
datetime d3=D'19.07.1980 12';              // равнозначно D'1980.07.19 12:00:00'
datetime d4=D'01.01.2004';                 // равнозначно D'01.01.2004 00:00:00'
datetime compilation_date=__DATE__;         // дата компиляции
datetime compilation_date_time=__DATETIME__; // дата и время компиляции
datetime compilation_time=__DATETIME__ - __DATE__; // время компиляции
//--- примеры объявлений, на которые будут получены предупреждения компилятора
datetime warning1=D'12:30:27';             // равнозначно D' [дата компиляции] 12:30:27'
datetime warning2=D'';                     // равнозначно __DATETIME__
```

**Смотри также**

[Структура даты](#), [Дата и время](#), [TimeToString](#), [StringToTime](#)

## Тип color

Тип **color** предназначен для хранения информации о цвете и занимает в памяти 4 байта. Первый байт не учитывается, остальные 3 байта содержат RGB-составляющие.

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 С 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 0x00BBGRR, 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
С'128,128,128'      // серый
С'0x00,0x00,0xFF'    // синий
//названия цветов
clrRed                // красный
clrYellow              // желтый
clrBlack               // черный
//--- целочисленные представления
0xFFFFFFF             // белый
16777215              // белый
0x008000               // зеленый
32768                 // зеленый
```

**Смотри также**

[Набор Web-цветов](#), [ColorToString](#), [StringToColor](#), [Приведение типов](#)

## Тип bool

Тип **bool** предназначен для хранения логических значений **true** (истина) или **false** (ложь), числовое представление которых 1 или 0 соответственно.

**Примеры:**

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

Внутреннее представление - целое число размером 1 байт. Необходимо отметить, что в логических выражениях допустимо использовать вместо типа **bool** другие целые или вещественные типы или выражения этих типов, компилятор не выдаст ошибки. В таком случае значение ноль будет интерпретировано как **false**, а все остальные значения как **true**.

**Примеры:**

```
int i=5;
double d=-2.5;
if(i)
    Print("i = ",i," и имеет значение true");
else
    Print("i = ",i," и имеет значение false");

if(d)
    Print("d = ",d," и имеет значение true");
else
    Print("d = ",d," и имеет значение false");

i=0;
if(i)
    Print("i = ",i," и имеет значение true");
else
    Print("i = ",i," и имеет значение false");

d=0.0;
if(d)
    Print("d = ",d," и имеет значение true");
else
    Print("d = ",d," и имеет значение false");

//--- результаты выполнения
//  i= 5 и имеет значение true
//  d= -2.5 и имеет значение true
//  i= 0 и имеет значение false
//  d= 0 и имеет значение false
```

**Смотри также**

[Логические операции, Приоритеты и порядок операций](#)

## Перечисления

Данные перечислимого типа `enum` относятся к некоторому ограниченному множеству данных. Определение перечислимого типа:

```
enum имя_перечислимого_типа
{
    список_значений
};
```

Список значений представляет из себя список идентификаторов именованных констант, разделенных запятыми.

### Пример:

```
enum months // перечисление именованных констант
{
    January,
    February,
    March,
    April,
    May,
    June,
    July,
    August,
    September,
    October,
    November,
    December
};
```

После объявления перечисления появляется новый целочисленный 4-байтовый тип данных. Объявление нового типа данных позволяет компилятору строго контролировать типы передаваемых параметров, так как перечисление вводит новые именованные константы. В приведенном примере именованная константа `January` имеет значение 0, `February` имеет значение 1, `December` имеет значение 11.

**Правило:** если именованной константе - члену перечисления явно не присвоено конкретное значение, то ее значение будет сформировано автоматически. Если это первый член перечисления, то будет присвоено значение 0. Для всех последующих членов значения будет вычисляться на основе значения предыдущего члена путем прибавления единицы.

### Пример:

```
enum intervals // перечисление именованных констант
{
    month=1,      // интервал в один месяц
    two_months,   // два месяца
    quarter,      // три месяца - квартал
    halfyear=6,    // полугодие
    year=12,       // год - 12 месяцев
};
```

## Примечания

- В отличие от C++, размер внутреннего представления перечислимого типа в MQL5 всегда составляет 4 байта. То есть, `sizeof(months)` вернет значение 4.
- В отличие от C++, в MQL5 нельзя объявить анонимное перечисление. То есть, после ключевого слова `enum` всегда должно быть указано уникальное имя.

Смотри также

[Приведение типов](#)

## Вещественные типы (double, float)

Вещественные типы (или типы с плавающей точкой) представляют значения, имеющие дробную часть. В языке MQL5 есть два типа для чисел с плавающей точкой. Способ представления вещественных чисел в машинной памяти определен стандартом IEEE 754 и не зависит от платформ, операционных систем и языков программирования.

Тип	Размер в байтах	Минимальное положительное значение	Максимальное значение	Аналог в C++
float	4	1.175494351e-38	3.402823466e+38	float
double	8	2.2250738585072014e-308	1.7976931348623158e+308	double

Имя `double` означает, что точность этих чисел вдвое превышает точность чисел типа `float`. В большинстве случаев тип `double` является наиболее удобным. Ограниченою точности чисел `float` во многих случаях попросту недостаточно. Причина, по которой тип `float` все еще используется, - экономия памяти при хранении (это важно для больших массивов вещественных чисел).

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;  
float c =0.0001;  
float d =16;
```

Существует научный способ записи вещественных констант, зачастую этот способ записи более компактный, чем традиционный.

### Пример:

Необходимо помнить, что вещественные числа хранятся в памяти компьютера с некоторой ограниченной точностью в двоичной системе счисления, в то время как общепринятой в использовании является десятичная система счисления. Поэтому многие числа, которые точно

записываются в десятичной системе, в двоичной системе можно записать только в виде бесконечной дроби.

Например, числа 0.3 и 0.7 представлены в компьютере бесконечными дробями, в то время как число 0.25 хранится точно, так как представляет из себя степень двойки.

В связи с этим, категорически не рекомендуется сравнивать между собой два вещественных числа на равенство, так как такое сравнение не является корректным.

**Пример:**

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

Если все же необходимо сравнить на равенство два вещественных числа, то можно сделать это двумя разными способами. Первый способ заключается в сравнении разницы между двумя числами с какой-то малой величиной, задающей точность сравнения.

**Пример:**

```
bool EqualDoubles(double d1,double d2,double epsilon)
{
    if(epsilon<0)
        epsilon=-epsilon;
//---
    if(d1-d2>epsilon)
        return false;
    if(d1-d2<-epsilon)
        return false;
//---
    return true;
}
void OnStart()
{
    double d_val=0.7;
    float f_val=0.7;
    if(EqualDoubles(d_val,f_val,0.0000000000000001))
        Print(d_val,"equals",f_val);
    else
```

```

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

```

Необходимо отметить, что значение параметра epsilon в приведенном примере не может быть меньше предопределенной константы DBL\_EPSILON. Значение этой константы 2.2204460492503131e-016. Для типа float соответствующая константа FLT\_EPSILON = 1.192092896e-07. Смысль этих значений таков, что это наименьшее значение, удовлетворяющее условию  $1.0 + \text{DBL\_EPSILON} \neq 1.0$  (для чисел типа float  $1.0 + \text{FLT\_EPSILON} \neq 1.0$ ).

Второй способ предполагает сравнивать нормализованную разность двух вещественных чисел с нулевым значением. Сравнивать разность нормализованных чисел с нулём бесполезно, так как в результате любой математической операции с нормализованными числами результат получается ненормализованным.

**Пример:**

```

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

```

В результате некоторых операций математического сопроцессора может получиться недействительное вещественное число, которое нельзя использовать в математических операциях и операциях сравнения, так как результат выполнения операций над недействительными вещественными числами неопределен. Например, при попытке вычислить [арксинус](#) от 2, результатом будет минус бесконечность.

**Пример:**

```

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

```

Кроме минус бесконечности существуют плюс бесконечность и NaN (не число). Чтобы определить, что данное число недействительно, можно использовать функцию [MathIsNaN\(\)](#). По стандарту IEEE они имеют специальное машинное представление. Например, плюс бесконечность для типа double имеет битовое представление 0x7FF0 0000 0000 0000.

## Примеры:

```

struct str1
{
    double d;
};

struct str2
{
    long l;
};

//--- начнем
str1 s1;
str2 s2;
//---

s1.d=MathArcsin(2.0);           // получим недействительное число -1.#IND
s2=s1;
printf("1. %f %I64X",s1.d,s2.l);

//---
s2.l=0xFFFF000000000000;        // недействительное число -1.#QNAN
s1=s2;
printf("2. %f %I64X",s1.d,s2.l);

//---
s2.l=0x7FF7000000000000;        // наибольшее нечисло SNaN
s1=s2;
printf("3. %f %I64X",s1.d,s2.l);

//---
s2.l=0x7FF8000000000000;        // наименьшее нечисло QNaN
s1=s2;
printf("4. %f %I64X",s1.d,s2.l);

//---
s2.l=0x7FFF000000000000;        // наибольшее нечисло QNaN
s1=s2;
printf("5. %f %I64X",s1.d,s2.l);

//---
s2.l=0x7FF0000000000000;        // плюс бесконечность 1.#INF и наименьшее нечисло SNaN
s1=s2;
printf("6. %f %I64X",s1.d,s2.l);

//---
s2.l=0xFFF0000000000000;        // минус бесконечность -1.#INF
s1=s2;
printf("7. %f %I64X",s1.d,s2.l);

//---
s2.l=0x8000000000000000;        // отрицательный ноль -0.0
s1=s2;
printf("8. %f %I64X",s1.d,s2.l);

//---
s2.l=0x3FE0000000000000;        // 0.5
s1=s2;

```

```
printf("9.    %f %I64X",s1.d,s2.l);
//---
s2.l=0x3FF0000000000000;      // 1.0
s1=s2;
printf("10.   %f %I64X",s1.d,s2.l);
//---
s2.l=0x7FEFFFFFFFFFFFFF;      // наибольшее нормализованное число (MAX_DBL)
s1=s2;
printf("11.   %.16e %I64X",s1.d,s2.l);
//---
s2.l=0x001000000000000;      // наименьшее положительное нормализованное (MIN_DBL)
s1=s2;
printf("12.   %.16e %.16I64X",s1.d,s2.l);
//---
s1.d=0.7;                     // покажем, что число 0.7 - бесконечная дробь
s2=s1;
printf("13.   %.16e %.16I64X",s1.d,s2.l);
/*
1. -1.#IND00 FFF8000000000000
2. -1.#QNAN0 FFFF000000000000
3. 1.#SNAN0 7FF7000000000000
4. 1.#QNAN0 7FF8000000000000
5. 1.#QNAN0 7FFF000000000000
6. 1.#INF00 7FF0000000000000
7. -1.#INF00 FFF0000000000000
8. -0.000000 8000000000000000
9. 0.500000 3FE0000000000000
10. 1.000000 3FF0000000000000
11. 1.7976931348623157e+308 7FEFFFFFFFFFFF
12. 2.2250738585072014e-308 0010000000000000
13. 6.999999999999996e-001 3FE6666666666666
*/
```

## Смотри также

[DoubleToString](#), [NormalizeDouble](#), [Константы числовых типов](#)

## Тип `string`

Тип `string` предназначен для хранения текстовых строк. Текстовая строка представляет собой последовательность символов в формате Unicode с завершающим нулем на конце. `string`-переменной может быть назначена строковая константа. Строковая константа представляет собой последовательность символов Unicode, заключенную в двойные кавычки: "Это строковая константа".

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 (\).

**Примеры:**

```
string svar="This is a character string";
string svar2=StringSubstr(svar,0,4);
Print("Символ копирайта\t\x00A9");
FileWrite(handle,"эта строка содержит символ перевода строки \n");
string MT5path="C:\\Program Files\\MetaTrader 5";
```

Длинные константные строки для удобства чтения исходного кода можно разбивать на части без операции сложения. Эти части при компиляции автоматически собираются в одну длинную строку:

```
///--- объявим длинную константную строку
string HTML_head="<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\""
                  " \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">\n"
                  "<html xmlns=\"http://www.w3.org/1999/xhtml\">\n"
                  "<head>\n"
                  " <meta http-equiv=\"Content-Type\" content=\"text/html; charset=ut
                  " <title>Trade Operations Report</title>\n"
                  "</head>";
///--- выведем в журнал константную строку
Print(HTML_head);
```

**Смотри также**

[Преобразование данных](#), [Строковые функции](#), [FileOpen](#), [FileReadString](#), [FileWriteString](#)

## Структуры, классы и интерфейсы

### Структуры

Структура является набором элементов произвольного типа (кроме типа [void](#)). Таким образом, структура объединяет логически связанные данные разных типов.

#### Объявление структуры

Структурный тип данных определяется следующим описанием:

```
struct имя_структурь
{
    описание_элементов
};
```

Имя структуры нельзя использовать в качестве идентификатора (имени переменной или функции). Следует иметь ввиду, что в MQL5 элементы структуры следуют непосредственно друг за другом без выравнивания. В языке C++ такое указание делается компилятору с помощью инструкции

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

Если требуется сделать иное выравнивание в структуре, необходимо использовать вспомогательные члены-"заполнители" нужных размеров.

**Пример:**

```
struct trade_settings
{
    uchar slippage;           // значение допустимого проскальзывания -размер 1 байт
    char reserved1;           // 1 байт пропуска
    short reserved2;          // 2 байта пропуска
    int reserved4;            // еще 4 байта пропуска. Обеспечили выравнивание на границу 8
    double take;              // значения цены фиксации прибыли
    double stop;               // значение цены защитного стопа
};
```

Такое описание выровненных структур необходимо только для передачи в импортированные dll-функции.

**Внимание:** данный пример иллюстрирует неправильно спроектированные данные. Лучше было бы сначала объявить данные `take` и `stop` большего размера типа [double](#), а затем объявить член `slippage` типа `uchar`. В этом случае внутреннее представление данных будет всегда одинаково независимо от значения, указанного в `#pragma pack()`.

Если структура содержит переменные типа [string](#) и/или [объект динамического массива](#), то компилятор назначает для такой структуры неявный конструктор, в котором производится обнуление всех членов структуры типа `string` и правильная инициализация для объекта динамического массива.

### Простые структуры

Структуры, которые не содержат строки, объекты класса, указатели и объекты динамических массивов, называются простыми структурами. Переменные простых структур, а также их массивы можно передавать в качестве параметров в [импортируемые](#) из DLL функции.

Копирование простых структур допускается только в двух случаях:

- если объекты принадлежат к одному типу структуры
- если объекты связаны между собой линией наследования, то есть одна структура является потомком другой структуры.

Покажем это на примерах, создадим пользовательскую структуру CustomMqlTick, идентичную по составу встроенной структуре [MqlTick](#). Попытки скопировать значение объекта MqlTick в объект типа CustomMqlTick компилятор не пропустит. [Прямое приведение](#) к нужному типу также вызовет ошибку компиляции:

```
//--- копировать простые структуры разных типов запрещено
my_tick1=last_tick; // компилятор здесь выдаст ошибку

//--- приводить структуры разного типа друг к другу тоже нельзя
my_tick1=(CustomMqlTick)last_tick;// компилятор здесь выдаст ошибку
```

Поэтому остается только один вариант - копировать значения членов структуры поэлементно. Но при этом разрешено копировать значения объектов одного и того же типа CustomMqlTick.

```
CustomMqlTick my_tick1,my_tick2;
//--- а вот так копировать объекты одной и той же структуры CustomMqlTick можно
my_tick2=my_tick1;

//--- создадим массив из объектов простой структуры CustomMqlTick и запишем в него
CustomMqlTick arr[2];
arr[0]=my_tick1;
arr[1]=my_tick2;
```

В качестве проверки вызывается функция [ArrayPrint\(\)](#) для вывода в журнал значения массива `arr[]`.

```
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- создадим такую же структуру, как встроенная MqlTick
struct CustomMqlTick
{
    datetime      time;          // Время последнего обновления цен
    double        bid;           // Текущая цена Bid
    double        ask;           // Текущая цена Ask
    double        last;          // Текущая цена последней сделки (Last)
    ulong         volume;        // Объем для текущей цены Last
    long          time_msc;     // Время последнего обновления цен в миллисекундах
    uint          flags;         // Флаги тиков
};
```

```

//--- получим значения последнего тика
MqlTick last_tick;
CustomMqlTick my_tick1,my_tick2;
//--- попытаемся скопировать данные из MqlTick в CustomMqlTick
if(SymbolInfoTick(Symbol(),last_tick))
{
    //--- копировать неродственные простые структуры запрещено
    //1. my_tick1=last_tick;                                // компилятор здесь выдаст ошибку

    //--- приводить неродственные структуры друг к другу тоже нельзя
    //2. my_tick1=(CustomMqlTick)last_tick;// компилятор здесь выдаст ошибку

    //--- поэтому копируем члены структуры поэлементно
    my_tick1.time=last_tick.time;
    my_tick1.bid=last_tick.bid;
    my_tick1.ask=last_tick.ask;
    my_tick1.volume=last_tick.volume;
    my_tick1.time_msc=last_tick.time_msc;
    my_tick1.flags=last_tick.flags;

    //--- а вот так копировать объекты одной и той же структуры CustomMqlTick можно
    my_tick2=my_tick1;

    //--- создадим массив из объектов простой структуры CustomMqlTick и запишем в него
    CustomMqlTick arr[2];
    arr[0]=my_tick1;
    arr[1]=my_tick2;
    ArrayPrint(arr);

    //--- пример вывода значений массива, содержащего объекты типа CustomMqlTick
    /*
        [time]      [bid]      [ask]      [last]      [volume]      [time_msc]      [flags]
    [0] 2017.05.29 15:04:37 1.11854 1.11863 +0.00000 1450000 1496070277157      2
    [1] 2017.05.29 15:04:37 1.11854 1.11863 +0.00000 1450000 1496070277157      2
    */
}
else
    Print("SymbolInfoTick() failed, error = ",GetLastError());
}

```

Второй пример показывает возможности копирования простых структур по линии наследования. Пусть у нас есть базовая структура `Animal`, от которой порождены наследованием структуры `Cat` и `Dog`. Мы можем копировать между собой объекты `Animal` и `Cat`, `Animal` и `Dog`, но не можем копировать между собой `Cat` и `Dog` - хотя оба являются потомками структуры `Animal`.

```

//--- структура для описания собак
struct Dog: Animal
{
    bool hunting;           // охотничья порода
};

```

```

//--- структура для описания кошек
struct Cat: Animal
{
    bool home; // домашняя порода
};

//--- создадим объекты дочерних структур
Dog dog;
Cat cat;

//--- можно копировать от предка к потомку (Animal ==> Dog)
dog=some_animal;
dog.swim=true; // собаки умеют плавать

//--- копировать объекты дочерних структур нельзя (Dog != Cat)
cat=dog; // компилятор здесь выдаст ошибку

```

Полный код примера:

```

//--- базовая структура для описания животных
struct Animal
{
    int head; // кол-во голов
    int legs; // кол-во ног
    int wings; // кол-во крыльев
    bool tail; // наличие хвоста
    bool fly; // летает
    bool swim; // плавает
    bool run; // бегает
};

//--- структура для описания собак
struct Dog: Animal
{
    bool hunting; // охотничья порода
};

//--- структура для описания кошек
struct Cat: Animal
{
    bool home; // домашняя порода
};

//-----+
//| Script program start function |
//-----+
void OnStart()
{
//--- создадим объект базового типа Animal и опишем его
    Animal some_animal;
    some_animal.head=1;
    some_animal.legs=4;
    some_animal.wings=0;
    some_animal.tail=true;
    some_animal.fly=false;
}

```

```

some_animal.swim=false;
some_animal.run=true;
//--- создадим объекты дочерних типов
Dog dog;
Cat cat;
//--- можно копировать от предка к потомку (Animal ==> Dog)
dog=some_animal;
dog.swim=true; // собаки умеют плавать
//--- копировать объекты дочерних структур нельзя (Dog != Cat)
//cat=dog; // компилятор здесь выдаст ошибку
//--- поэтому можно копировать только поэлементно
cat.head=dog.head;
cat.legs=dog.legs;
cat.wings=dog.wings;
cat.tail=dog.tail;
cat.fly=dog.fly;
cat.swim=false; // кошки не умеют плавать
//--- копировать значения от потомка к предку можно
Animal elephant;
elephant=cat;
elephant.run=false;// слоны не умеют бегать
elephant.swim=true;// слоны плавают
//--- создадим массив
Animal animals[4];
animals[0]=some_animal;
animals[1]=dog;
animals[2]=cat;
animals[3]=elephant;
//--- выведем на печать
ArrayPrint(animals);
//--- результат выполнения
/*
      [head] [legs] [wings] [tail] [fly] [swim] [run]
[0]      1      4      0  true false  false  true
[1]      1      4      0  true false   true  true
[2]      1      4      0  true false  false false
[3]      1      4      0  true false  true false
*/
}
}

```

Другим способом копировать простые типы является использование объединения, для этого объекты этих структур должны являться членами одного и того же объединения - смотрите пример в [union](#).

## Доступ к членам структуры

Имя структуры является новым типом данных и позволяет объявлять переменные этого типа. Структура может быть объявлена только один раз в пределах проекта. Доступ к членам структур производится при помощи [операции точка](#) (.) .

**Пример:**

```
struct trade_settings
{
    double take;           // значения цены фиксации прибыли
    double stop;           // значение цены защитного стопа
    uchar slippage;        // значение допустимого проскальзывания
};

//--- создали и проинициализировали переменную типа trade_settings
trade_settings my_set={0.0,0.0,5};
if (input_TP>0) my_set.take=input_TP;
```

## pack для выравнивания полей структур и классов

Специальный атрибут **pack** позволяет задать выравнивание полей структуры или класса.

```
pack([n])
```

где n - одно из следующих значений 1,2,4,8 или 16. Может отсутствовать.

**Пример:**

```
struct pack(sizeof(long)) MyStruct
{
    // члены структуры будут выровнены на границу 8 байт
};

или

struct MyStruct pack(sizeof(long))
{
    // члены структуры будут выровнены на границу 8 байт
};
```

По умолчанию для структур используется **pack(1)**. Это означает, что в памяти члены структуры располагаются друг за другом и размер структуры равен сумме размеров её членов.

**Пример:**

```
//+-----+
//| Script program start function          |
//+-----+
void OnStart()
{
//--- простая структура без выравнивания
    struct Simple_Structure
    {
        char      c; // sizeof(char)=1
        short     s; // sizeof(short)=2
        int       i; // sizeof(int)=4
        double    d; // sizeof(double)=8
    };
//--- объявим экземпляр простой структуры
    Simple_Structure s;
```

```

//--- выведем размер каждого члена структуры
Print("sizeof(s.c)=", sizeof(s.c));
Print("sizeof(s.s)=", sizeof(s.s));
Print("sizeof(s.i)=", sizeof(s.i));
Print("sizeof(s.d)=", sizeof(s.d));
//--- убедимся, что размер POD-структуре равен сумме размеров её членов
Print("sizeof(simple_structure)=", sizeof(simple_structure));
/*
Результат:
sizeof(s.c)=1
sizeof(s.s)=2
sizeof(s.i)=4
sizeof(s.d)=8
sizeof(simple_structure)=15
*/
}

```

Выравнивание полей структуры может понадобится при обмене данными со сторонними библиотеками (\*.DLL), в которых такое выравнивание применяется.

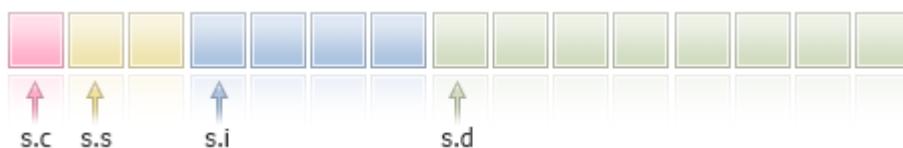
Покажем на примерах, как работает выравнивание. Возьмем структуру из четырех членов без выравнивания.

```

//--- простая структура без выравнивания
struct Simple_Structure pack() // размер не указан, будет выравнивание на границу 4 байт
{
    char      c; // sizeof(char)=1
    short     s; // sizeof(short)=2
    int       i; // sizeof(int)=4
    double   d; // sizeof(double)=8
};
//--- объявим экземпляр простой структуры
Simple_Structure s;

```

Поля структуры будут располагаться в памяти друг за другом, согласно порядку [объявления и размеру типа](#). Размер структуры равен 15, смещение к полям структуры в массивах будет неопределённым.



Объявим теперь эту же структуру с выравниванием в 4 байта и запустим код.

```

//-----+
//| Script program start function |
//-----+
void OnStart()
{
//--- простая структура с выравниванием в 4 байта
}

```

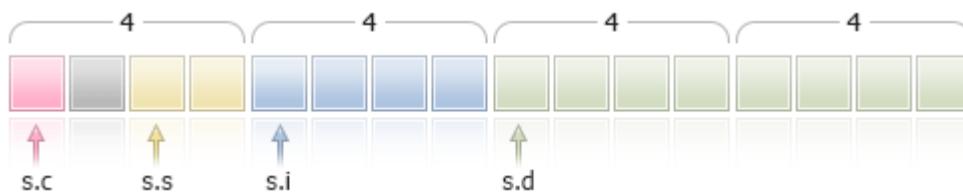
```

struct Simple_Structure pack(4)
{
    char          c; // sizeof(char)=1
    short        s; // sizeof(short)=2
    int          i; // sizeof(int)=4
    double       d; // sizeof(double)=8
};

//--- объявим экземпляр простой структуры
Simple_Structure s;
//--- выведем размер каждого члена структуры
Print("sizeof(s.c)=", sizeof(s.c));
Print("sizeof(s.s)=", sizeof(s.s));
Print("sizeof(s.i)=", sizeof(s.i));
Print("sizeof(s.d)=", sizeof(s.d));
//--- убедимся, что размер POD-структуре теперь не равен сумме размеров её членов
Print("sizeof(simple_structure)=", sizeof(simple_structure));
/*
Результат:
sizeof(s.c)=1
sizeof(s.s)=2
sizeof(s.i)=4
sizeof(s.d)=8
sizeof(simple_structure)=16 // размер структуры изменился
*/
}

```

Размер структуры изменился таким образом, чтобы все члены размером 4 байта или больше имели смещение от начала структуры кратное 4 байтам. Члены меньшего размера будут выравниваться на границу своего размера (например 2 для short). Вот как это выглядит, добавленный байт показан серым цветом.



В данном случае после члена `s.c` добавлен 1 байт, чтобы поле `s.s` (`sizeof(short)==2`) имело границу 2 байта (выравнивание для типа `short`).

Смещение к началу структуры в массиве также будет выравнено на границу 4 байт, т.е. для `Simple_Structure arr[]`, адреса элементов `a[0], a[1], a[n]` будут кратными 4 байтам.

Рассмотрим еще две структуры, которые состоят из одинаковых типов с выравниванием на 4 байта, но при этом порядок следования членов отличается. В первой структуре члены располагаются по возрастанию размера типа.

```

//+-----+
//| Script program start function
//+-----+

```

```

void OnStart()
{
//--- простая структура с выравниванием на границу 4 байта
struct CharShortInt pack(4)
{
    char      c; // sizeof(char)=1
    short     s; // sizeof(short)=2
    int       i; // sizeof(double)=4
};

//--- объявим экземпляр простой структуры
CharShortInt ch_sh_in;
//--- выведем размер каждого члена структуры
Print("sizeof(ch_sh_in.c)=",sizeof(ch_sh_in.c));
Print("sizeof(ch_sh_in.s)=",sizeof(ch_sh_in.s));
Print("sizeof(ch_sh_in.i)=",sizeof(ch_sh_in.i));

//--- убедимся, что размер POD-структурь равен сумме размеров её членов
Print("sizeof(CharShortInt)=",sizeof(CharShortInt));
/*
Результат:
sizeof(ch_sh_in.c)=1
sizeof(ch_sh_in.s)=2
sizeof(ch_sh_in.i)=4
sizeof(CharShortInt)=8
*/
}
}

```

Как видим, размер структуры равен 8 и состоит из двух блоков по 4 байта. В первом блоке размещаются поля с типами `char` и `short`, во втором блоке находится поле с типом `int`.



Теперь из первой структуры сделаем вторую, которая отличается только порядком следования полей - переставим член типа `short` в конец.

```

//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- простая структура с выравниванием на границу 4 байта
struct CharIntShort pack(4)
{
    char      c; // sizeof(char)=1
    int       i; // sizeof(double)=4
    short     s; // sizeof(short)=2
};

//--- объявим экземпляр простой структуры

```

```

CharIntShort ch_in_sh;
//--- выводем размер каждого члена структуры
Print("sizeof(ch_in_sh.c)=", sizeof(ch_in_sh.c));
Print("sizeof(ch_in_sh.i)=", sizeof(ch_in_sh.i));
Print("sizeof(ch_in_sh.s)=", sizeof(ch_in_sh.s));
//--- убедимся, что размер POD-структурь равен сумме размеров её членов
Print("sizeof(CharIntShort)=", sizeof(CharIntShort));
/*
Результат:
sizeof(ch_in_sh.c)=1
sizeof(ch_in_sh.i)=4
sizeof(ch_in_sh.s)=2
sizeof(CharIntShort)=12
*/
}

```

Хотя сам состав структуры практически не изменился, но изменение порядка членов привело к увеличению размера самой структуры.



При наследовании также необходимо учитывать выравнивание. Покажем на примере простой структуры Parent, которая имеет только один член типа char. Размер такой структуры без выравнивания равен 1.

```

struct Parent
{
    char           c;      // sizeof(char)=1
};

```

Создаём дочерний класс Children с добавлением члена типа short (sizeof(short)=2).

```

struct Children pack(2) : Parent
{
    short          s;      // sizeof(short)=2
};

```

В результате при установке выравнивания в 2 байта размер структуры будет равен 4, хотя размер самих членов в ней равен 3. В данном примере под родительский класс Parent будет выделено 2 байта, чтобы доступ к полю short дочернего класса был выровнен на 2 байта.

Знание того, как распределяется память под члены структуры необходимо, если MQL5-программа взаимодействует со сторонними данными посредством записи/чтения на уровне файлов или потоков.

В [Стандартной библиотеке](#) в каталоге MQL5\Include\WinAPI представлены функции для работы с функциями WinAPI. Эти функции используют структуры с заданным выравниванием для тех случаев, когда это требуется для работы с WinAPI.

**offsetof** - это специальная команда, которая непосредственно связана в атрибутом [pack](#). Она позволяет получить смещение члена от начала структуры.

```
//--- объявим переменную типа Children
Children child;
//--- узнаем смещения от начала структуры
Print("offsetof(Children,c)=", offsetof(Children,c));
Print("offsetof(child.s)=", offsetof(child.s));
/*
Результат:
offsetof(Children,c)=0
offsetof(child.s)=2
*/
```

## Модификатор final

Наличие модификатора **final** при объявлении структуры запрещает дальнейшее наследование от нее. Если структура такова, что нет необходимости вносить в нее дальнейшие изменения, или изменения не допустимы по соображениям безопасности, объягите ее с модификатором **final**. При этом все члены структуры будут также неявно считаться **final**.

```
struct settings final
{
    //--- тело структуры
};

struct trade_settings : public settings
{
    //--- тело структуры
};
```

При попытке наследования от структуры с модификатором **final**, как показано в примере выше, компилятор выдаст ошибку:

```
cannot inherit from 'settings' as it has been declared as 'final'
see declaration of 'settings'
```

## Классы

Классы имеют ряд отличий от структур:

- в объявлении используется ключевое слово **class**;
- по умолчанию все члены класса имеют спецификатор доступа **private**, если не указано иное. Члены-данные структуры по умолчанию имеют тип доступа **public**, если не указано иное;
- объекты классов всегда имеют таблицу [виртуальных функций](#), даже если в классе не объявлено ни одной виртуальной функции. Структуры не могут иметь виртуальных функций;
- к объектам класса можно применять оператор [new](#), к структурам этот оператор применять нельзя;
- классы могут [наследоваться](#) только от классов, структуры могут наследоваться только от структур.

Классы и структуры могут иметь явный конструктор и деструктор. В случае если явно определен конструктор, инициализация переменной типа структуры или класса при помощи инициализирующей последовательности невозможна.

**Пример:**

```
struct trade_settings
{
    double take;           // значения цены фиксации прибыли
    double stop;           // значение цены защитного стопа
    uchar slippage;        // значение допустимого проскальзывания
    //--- конструктор
    trade_settings() { take=0.0; stop=0.0; slippage=5; }
    //--- деструктор
    ~trade_settings() { Print("Это конец"); }
};

//--- компилятор выдаст ошибку с сообщением о невозможности инициализации
trade_settings my_set={0.0,0.0,5};
```

## Конструкторы и деструкторы

Конструктор - это специальная функция, которая вызывается автоматически при создании объекта структуры или класса и обычно используется для [инициализации](#) членов класса. Далее мы будем говорить только о классах, при этом все сказанное относится и к структурам, если не оговорено иное. Имя конструктора должно совпадать с именем класса. Конструктор не имеет возвращаемого типа (можно указать тип [void](#)).

Определенные члены класса - [строки](#), [динамические массивы](#) и объекты, требующие инициализации - в любом случае будут проинициализированы, независимо от наличия конструктора.

Каждый класс может иметь несколько конструкторов, отличающихся по количеству параметров и спискам инициализации. Конструктор, требующий указания параметров, называется параметрическим конструктором.

Конструктор, не имеющий параметров, называется **конструктором по умолчанию**. Если в классе не объявлен ни один конструктор, то компилятор сам создаст конструктор по умолчанию при компиляции.

```
//+-----+
//| Класс для работы с датой |
//+-----+
class MyDateClass
{
private:
    int         m_year;      // год
    int         m_month;     // месяц
    int         m_day;       // день месяца
    int         m_hour;      // час в сутках
    int         m_minute;    // минуты
    int         m_second;    // секунды
```

```

public:
    //--- конструктор по умолчанию
    MyDateClass(void);
    //--- конструктор с параметрами
    MyDateClass(int h,int m,int s);
}

```

Конструктор можно объявить в описании класса, а затем определить его тело. Например, вот так могут быть определены два конструктора класса `MyDateClass`:

```

//+-----+
// | Конструктор по умолчанию
//+-----+
MyDateClass::MyDateClass(void)
{
//---

    MqlDateTime mdt;
    datetime t=TimeCurrent(mdt);
    m_year=mdt.year;
    m_month=mdt.mon;
    m_day=mdt.day;
    m_hour=mdt.hour;
    m_minute=mdt.min;
    m_second=mdt.sec;
    Print(__FUNCTION__);
}

//+-----+
// | Конструктор с параметрами
//+-----+
MyDateClass::MyDateClass(int h,int m,int s)
{
    MqlDateTime mdt;
    datetime t=TimeCurrent(mdt);
    m_year=mdt.year;
    m_month=mdt.mon;
    m_day=mdt.day;
    m_hour=h;
    m_minute=m;
    m_second=s;
    Print(__FUNCTION__);
}

```

В [конструкторе по умолчанию](#) заполняются все члены класса с помощью функции `TimeCurrent()`, в конструкторе с параметрами заполняются только значения часа. Остальные члены класса (`m_year`, `m_month` и `m_day`) будут проинициализированы автоматически текущей датой.

Конструктор по умолчанию имеет специальное назначение при инициализации массива объектов своего класса. Конструктор, все параметры которого имеют значения по умолчанию, **не является** конструктором по умолчанию. Покажем это на примере:

```

//+-----+
//| Класс с конструктором по умолчанию           |
//+-----+
class CFoo
{
    datetime      m_call_time;      // время последнего обращения к объекту
public:
    //--- конструктор с параметром, имеющим значение по умолчанию, не является конструктором
    CFoo(const datetime t=0){m_call_time=t;};
    //--- конструктор копирования
    CFoo(const CFoo &foo){m_call_time=foo.m_call_time;};

    string ToString(){return(ToString(m_call_time,TIME_DATE|TIME_SECONDS));};
};

//+-----+
//| Script program start function                 |
//+-----+
void OnStart()
{
    // CFoo foo; // такой вариант использовать нельзя - конструктор по умолчанию не задан
    //--- допустимые варианты создания объекта CFoo
    CFoo foo1(TimeCurrent());          // явный вызов параметрического конструктора
    CFoo foo2();                      // явный вызов параметрического конструктора с параметром
    CFoo foo3=D'2009.09.09';          // неявный вызов параметрического конструктора
    CFoo foo40(foo1);                // явный вызов конструктора копирования
    CFoo foo41=foo1;                  // неявный вызов конструктора копирования
    CFoo foo5;                        // явный вызов конструктора по умолчанию (если конструктор по умолчанию не определен, то вызывается параметрический конструктор с параметром)
    //--- допустимые варианты получения указателей CFoo
    CFoo *pfoo6=new CFoo();           // динамическое создание объекта и получение указателя
    CFoo *pfoo7=new CFoo(TimeCurrent()); // еще один вариант динамического создания объекта
    CFoo *pfoo8=GetPointer(foo1);     // теперь pfoo8 указывает на объект foo1
    CFoo *pfoo9=pfoo7;               // pfoo9 и pfoo7 указывают на один и тот же объект
    // CFoo foo_array[3];             // такой вариант использовать нельзя - конструктор по умолчанию не определен
    //--- выведем значения m_call_time
    Print("foo1.m_call_time=",foo1.ToString());
    Print("foo2.m_call_time=",foo2.ToString());
    Print("foo3.m_call_time=",foo3.ToString());
    Print("foo4.m_call_time=",foo4.ToString());
    Print("foo5.m_call_time=",foo5.ToString());
    Print("pfoo6.m_call_time=",pfoo6.ToString());
    Print("pfoo7.m_call_time=",pfoo7.ToString());
    Print("pfoo8.m_call_time=",pfoo8.ToString());
    Print("pfoo9.m_call_time=",pfoo9.ToString());
    //--- удалим динамически созданные объекты

```

```

    delete pfoo6;
    delete pfoo7;
    //delete pfoo8; // удалять pfoo8 явно не нужно, так как он указывает на автоматически
    //delete pfoo9; // удалять pfoo9 явно не нужно, так как он указывает на тот же обьект
}

```

Если раскомментировать в этом примере строки

```
//CFoo foo_array[3]; // такой вариант использовать нельзя – конструктор по умолчанию
```

или

```
//CFoo foo_dyn_array[]; // такой вариант использовать нельзя – конструктор по умолчанию
```

то компилятор выдаст на них ошибку "default constructor is not defined".

Если класс имеет конструктор, объявленный пользователем, то конструктор по умолчанию не будет сгенерирован компилятором. Это означает, что если в классе объявлен конструктор с параметрами, но не объявлен конструктор по умолчанию, то нельзя объявлять массивы объектов этого класса. Вот на таком скрипте компилятор сообщит об ошибке:

```

//+-----+
//| Класс без конструктора по умолчанию |
//+-----+
class CFoo
{
    string m_name;
public:
    CFoo(string name) { m_name=name; }
};

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- при компиляции получим ошибку "default constructor is not defined"
    CFoo badFoo[5];
}

```

В данном примере класс CFoo имеет объявленный параметрический конструктор - в таких случаях компилятор при компиляции не создает автоматически конструктор по умолчанию. В то же время при объявлении массива объектов предполагается, что все объекты должны быть [созданы и инициализированы автоматически](#). При автоматической инициализации объекта требуется вызвать конструктор по умолчанию, но так как конструктор по умолчанию явно не объявлен и не сгенерирован автоматически компилятором, то создание такого объекта невозможно. Именно по этой причине компилятор выдает ошибку еще на этапе компиляции.

Существует специальный синтаксис для инициализации объекта с помощью конструктора. Инициализаторы конструктора (специальные конструкции для инициализации) для членов структуры или класса могут быть заданы в списке инициализации.

Список инициализации - это список инициализаторов, разделенных запятыми, который идет после двоеточия за [списком параметров](#) конструктора и предшествует [телу](#) (идет перед открывающей фигурной скобкой). Есть несколько требований:

- списки инициализации можно использовать только в [конструкторах](#);
- в списке инициализации нельзя инициализировать [члены родителей](#);
- после списка инициализации должно идти [определение](#) (реализация) функции.

Покажем пример нескольких конструкторов для инициализации членов класса.

```
//+-----+
//| Класс для хранения фамилии и имени персонажа
//+-----+
class CPerson
{
    string          m_first_name;      // имя
    string          m_second_name;     // фамилия

public:
    //--- пустой конструктор по умолчанию
    CPerson() { Print(__FUNCTION__); }

    //--- параметрический конструктор
    CPerson(string full_name);

    //--- конструктор со списком инициализации
    CPerson(string surname, string name) : m_second_name(surname), m_fi
    void PrintName() { PrintFormat("Name=%s Surname=%s", m_first_name, m_se
    };

    //+-----+
    //|
    //+-----+
    CPerson::CPerson(string full_name)
    {
        int pos=StringFind(full_name, " ");
        if(pos>=0)
        {
            m_first_name=StringSubstr(full_name, 0, pos);
            m_second_name=StringSubstr(full_name, pos+1);
        }
    }

    //+-----+
    //| Script program start function
    //+-----+
    void OnStart()
    {
        //--- получим ошибку "default constructor is not defined"
        CPerson people[5];
        CPerson Tom="Tom Sawyer";                                // Том Сойер
        CPerson Huck("Huckleberry", "Finn");                   // Гекльберри Финн
        CPerson *Pooh = new CPerson("Winnie", "Pooh");         // Винни Пух
        //--- выведем значения
    }
}
```

```

Tom.PrintName();
Huck.PrintName();
Pooh.PrintName();

//--- удалим динамически созданный объект
delete Pooh;
}

```

В данном случае класс **CPerson** имеет три конструктора:

1. явный [конструктор по умолчанию](#), который позволяет создавать массив объектов данного класса;
2. конструктор с одним параметром, который принимает в качестве параметра полное имя и разделяет его на имя и фамилию по найденному пробелу;
3. конструктор с двумя параметрами, который содержит [список инициализации](#). Инициализаторы - **m\_second\_name(surname)** и **m\_first\_name(name)**.

Обратите внимание, как инициализация с помощью списка заменила присваивание. Отдельные члены должны быть инициализированы как:

```
член_класса (список выражений)
```

В списке инициализации члены могут идти в любом порядке, но при этом все члены класса будут инициализироваться согласно порядку их объявления. Это означает, что в третьем конструкторе сначала будет инициализирован член **m\_first\_name**, так как он объявлен первым, и только после него будет инициализирован член **m\_second\_name**. Это необходимо учитывать в тех случаях, когда инициализация одних членов класса зависит от значений в других членах класса.

Если в базовом классе не объявлен конструктор по умолчанию и при этом объявлен один или несколько конструкторов с параметрами, то нужно обязательно вызвать один из конструкторов базового класса в списке инициализации. Он идет через запятую как обычные члены списка и будет вызван в первую очередь при инициализации объекта независимо от местоположения в списке инициализации.

```

//+-----+
//| Базовый класс
//+-----+
class CFoo
{
    string          m_name;
public:
    //--- конструктор со списком инициализации
    CFoo(string name) : m_name(name) { Print(m_name); }
};

//+-----+
//| Потомок класса CFoo
//+-----+
class CBar : CFoo
{
    CFoo           m_member;      // член класса является объектом предка
public:

```

```

//--- конструктор по умолчанию в списке инициализации вызывает конструктор предка
CBar(): m_member(_Symbol), CFoo("CBAR") {Print(__FUNCTION__);}

};

//-----+
//| Script program start function
//+-----+
void OnStart()
{
    CBar bar;
}

```

В приведенном примере при создании объекта bar будет вызван конструктор по умолчанию CBar(), в котором сначала вызывается конструктор для предка CFoo, а затем конструктор для члена класса m\_member.

Деструктор - это специальная функция, которая вызывается автоматически при уничтожении объекта класса. Имя деструктора записывается как имя класса с тильдой (~). Строки, динамические массивы и объекты, требующие deinициализации, в любом случае будут deinициализированы независимо от наличия деструктора. При наличии деструктора, эти действия будут произведены после вызова деструктора.

Деструкторы всегда являются [виртуальными](#), независимо от того, объявлены они с ключевым слово **virtual** или нет.

## Определение методов классов

Функции-методы класса могут быть определены как внутри класса, так и за пределами объявления класса. Если метод определяется внутри класса, то его тело следует непосредственно после объявления метода.

**Пример:**

```

class CTetrisShape
{
protected:
    int          m_type;
    int          m_xpos;
    int          m_ypos;
    int          m_xsize;
    int          m_ysize;
    int          m_prev_turn;
    int          m_turn;
    int          m_right_border;
public:
    void        CTetrisShape();
    void        SetRightBorder(int border) { m_right_border=border; }
    void        SetYPos(int ypos)           { m_ypos=ypos; }
    void        SetXPos(int xpos)          { m_xpos=xpos; }
    int         GetYPos()                { return(m_ypos); }
    int        GetXPos()                { return(m_xpos); }
    int         GetYSize()               { return(m_ysize); }

```

```

int          GetXSize()           { return(m_xsize); }
int          GetType()            { return(m_type); }
void         Left()               { m_xpos-=SHAPE_SIZE; }
void         Right()              { m_xpos+=SHAPE_SIZE; }
void         Rotate()              { m_prev_turn=m_turn; if(++m_turn>3) r
virtual void Draw()               { return; }
virtual bool CheckDown(int& pad_array[]);
virtual bool CheckLeft(int& side_row[]);
virtual bool CheckRight(int& side_row[]);
};

}

```

Функции с SetRightBorder(int border) по Draw() объявлены и определены прямо внутри класса CTetrisShape.

Конструктор CTetrisShape() и методы CheckDown(int& pad\_array[]), CheckLeft(int& side\_row[]) и CheckRight(int& side\_row[]) только объявлены внутри класса, но пока не определены. Определения этих функций должны следовать далее по коду. Для того чтобы определить метод вне класса используется [операция разрешения контекста](#), в качестве контекста используется имя класса.

**Пример:**

```

//+-----+
//| Конструктор базового класса |
//+-----+
void CTetrisShape::CTetrisShape()
{
    m_type=0;
    m_ypos=0;
    m_xpos=0;
    m_xsize=SHAPE_SIZE;
    m_ysize=SHAPE_SIZE;
    m_prev_turn=0;
    m_turn=0;
    m_right_border=0;
}

//+-----+
//| Проверка возможности двигаться вниз (для палки и куба) |
//+-----+
bool CTetrisShape::CheckDown(int& pad_array[])
{
    int i,xsize=m_xsize/SHAPE_SIZE;
    //---
    for(i=0; i<xsize; i++)
    {
        if(m_ypos+m_ysize>=pad_array[i]) return(false);
    }
    //---
    return(true);
}

```

## Модификаторы доступа public, protected и private

При разработке нового класса рекомендуется ограничивать доступ к членам извне. Для этих целей используются ключевые слова **private** или **protected**. Доступ в этом случае к скрытым данным может осуществляться только из функций-методов этого же класса. Если использовано ключевое слово **protected**, то доступ к скрытым данным может осуществляться и из методов классов - наследников этого класса. Точно таким же образом может ограничиваться доступ и к функциям-методам класса.

Если необходимо полностью открыть доступ к членам и/или методам класса, то используется ключевое слово **public**.

**Пример:**

```
class CTetrisField
{
private:
    int             m_score;                      // счёт
    int             m_ypos;                        // текущее положение фигуры
    int             m_field[FIELD_HEIGHT][FIELD_WIDTH]; // матрица стакана
    int             m_rows[FIELD_HEIGHT];           // нумерация рядов стакана
    int             m_last_row;                     // последний свободный ряд
    CTetrisShape   *m_shape;                       // тетрисная фигура
    bool            m_bover;                        // игра закончена
public:
    void            CTetrisField() { m_shape=NULL; m_bover=false; }
    void            Init();
    void            Deinit();
    void            Down();
    void            Left();
    void            Right();
    void            Rotate();
    void            Drop();
private:
    void            NewShape();
    void            CheckAndDeleteRows();
    void            LabelOver();
};
```

Любые члены и методы класса, объявленные после спецификатора **public:** (и до следующего спецификатора доступа), доступны при любом обращении программы к объекту этого класса. В данном примере это следующие члены: функции `CTetrisField()`, `Init()`, `Deinit()`, `Down()`, `Left()`, `Right()`, `Rotate()` и `Drop()`.

Любые члены класса, объявленные после спецификатора доступа к элементам **private:** (и до следующего спецификатора доступа), доступны только функциям-членам этого класса. Спецификаторы доступа к элементам всегда заканчиваются двоеточием (:) и могут появляться в определении класса много раз.

Доступ к членам базового класса может переопределяться при наследовании в производных классах.

## Модификатор final

Наличие модификатора `final` при объявлении класса запрещает дальнейшее наследование от него. Если интерфейс класса таков, что нет необходимости вносить в него дальнейшие изменения, или изменения не допустимы по соображениям безопасности, объягите класс с модификатором `final`. При этом все методы класса будут также неявно считаться `final`.

```
class CFoo final
{
    //--- тело класса
};

class CBar : public CFoo
{
    //--- тело класса
};
```

При попытке наследования от класса с модификатором `final`, как показано в примере выше, компилятор выдаст ошибку:

```
cannot inherit from 'CFoo' as it has been declared as 'final'
see declaration of 'CFoo'
```

## Объединение (union)

Объединение - это особый тип данных, который состоит из нескольких переменных, разделяющих одну и ту же область памяти. Следовательно, объединение обеспечивает возможность интерпретации одной и той же последовательности битов двумя (или более) различными способами. Объявление объединения подобно объявлению [структуре](#) и начинается с ключевого слова `union`.

```
union LongDouble
{
    long long_value;
    double double_value;
};
```

Но в отличие от структуры, разные члены объединения относятся к одному и тому же участку памяти. В данном примере объявлено объединение `LongDouble`, в котором значение типа `long` и значение типа `double` разделяют одну и ту же область памяти. Важно понимать - невозможно сделать так, чтобы это объединение хранило одновременно целочисленное значение `long` и вещественное `double` (как это было бы в структуре), поскольку переменные `long_value` и `double_value` накладываются (в памяти) друг на друга. Но зато MQL5-программа в любой момент может обрабатывать информацию, содержащуюся в этом объединении, как целочисленное значение (`long`) или как вещественное (`double`). Следовательно, объединение позволяет получить два (или больше) варианта представления одной и той же последовательности данных.

При объявлении объединения компилятор автоматически выделяет область памяти, достаточную для хранения в объединении переменных самого большого по [объему типа](#). Для доступа к элементу объединения используется тот же синтаксис, как и для структур - [оператор "точка"](#).

```
union LongDouble
{
    long long_value;
```

```

    double double_value;
};

//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
//---

    LongDouble lb;

//--- получим недействительное число -nan(ind) и выведем его
    lb.double_value=MathArcsin(2.0);
    printf("1. double=%f           integer=%I64X",lb.double_value,lb.long_value);

//--- наибольшее нормализованное число (DBL_MAX)
    lb.long_value=0x7FEFFFFFFFFFFFFF;
    printf("2. double=%.16e  integer=%I64X",lb.double_value,lb.long_value);

//--- наименьшее положительное нормализованное (DBL_MIN)
    lb.long_value=0x0010000000000000;
    printf("3. double=%.16e  integer=%I64X",lb.double_value,lb.long_value);
}

/* Результат выполнения
1. double=-nan(ind)           integer=FFF8000000000000
2. double=1.7976931348623157e+308  integer=7FEFFFFFFFFFFF
3. double=2.2250738585072014e-308  integer=0010000000000000
*/

```

Поскольку объединения позволяют программе интерпретировать одни и те же данные в памяти по-разному, они часто используются в случаях, когда требуется необычное [преобразование типов](#).

Объединения не могут участвовать в [наследовании](#), а также они не могут иметь [статических членов](#) по определению. В остальном *union* ведёт себя как структура, у которой все члены имеют нулевое смещение. При этом членами объединения не могут быть следующие типы:

- [динамические массивы](#)
- [строки](#)
- [указатели на объекты и функции](#)
- [объекты классов](#)
- [объекты структур, имеющие конструкторы или деструкторы](#)
- [объекты структур, имеющие в себе члены из пунктов 1-5](#)

Также как и классы, объединение может иметь конструкторы и деструкторы, а также и методы. По умолчанию члены объединения имеют тип доступа [public](#), для создания закрытых элементов необходимо использовать ключевое слово [private](#). Все эти возможности представлены в примере, который показывает как преобразовать цвет, имеющий тип [color](#), в представление ARGB, как это делает функция [ColorToARGB\(\)](#).

```

//+-----+
//| Объединение для конвертации color(BGR) в представление ARGB |
//+-----+

union ARGB
{

```

```

uchar           argb[4];
color          clr;
//--- конструкторы
ARGB(color col,uchar a=0){Color(col,a);}
~ARGB() {};
//--- публичные методы
public:
uchar  Alpha(){return(argb[3]);};
void   Alpha(const uchar alpha){argb[3]=alpha;};
color  Color(){ return(color(clr));};
//--- закрытые методы
private:
//+-----+
//| установка цвета и значения альфа-канала |
//+-----+
void  Color(color col,uchar alpha)
{
//--- установим цвет в член clr
clr=col;
//--- установим значение компонента Alpha - уровня непрозрачности
argb[3]=alpha;
//--- переставим местами байты компонент R и B (Red и Blue)
uchar t=argb[0];argb[0]=argb[2];argb[2]=t;
};
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- значение 0x55 означает 55/255=21.6 % (0% - полностью прозрачный)
uchar alpha=0x55;
//--- тип color имеет представление 0x00BBGRR
color test_color=clrDarkOrange;
//--- сюда будем принимать значения байтов из объединения ARGB
uchar argb[];
PrintFormat("0x%.8X - так выглядит тип color для %s, BGR=(%s)",
           test_color,ColorToString(test_color,true),ColorToString(test_color));
//--- тип ARGB представлен как 0x00RRGGBB, переставлены местами RR и BB компоненты
ARGB argb_color(test_color);
//--- скопируем массив байтов
ArrayCopy(argb,argb_color.argb);
//--- посмотрим как выглядит в представлении ARGB
PrintFormat("0x%.8X - представление ARGB с альфа-каналом=0x%.2x, ARGB=(%d,%d,%d,%d)",
           argb_color.clr,argb_color.Alpha(),argb[3],argb[2],argb[1],argb[0]);
//--- добавим значение непрозрачности
argb_color.Alpha(alpha);
//--- попробуем вывести ARGB как тип color
Print("ARGB как color=(",argb_color.clr,") альфа-канал=",argb_color.Alpha());

```

```

//--- скопируем массив байтов
ArrayCopy(argb,argb_color.argb);
//--- а вот как выглядит в представлении ARGB
PrintFormat("0x%.8X - представление ARGB с альфа-каналом=0x%.2x, ARGB=(%d,%d,%d,%d)
            argb_color.clr,argb_color.Alpha(),argb[3],argb[2],argb[1],argb[0]);
//--- сверим с тем, что выдает функция ColorToARGB()
PrintFormat("0x%.8X - результат ColorToARGB(%s,0x%.2x)",ColorToARGB(test_color,alpha
            ColorToString(test_color,true),alpha);
}
/* Результат выполнения
0x000008CFF - так выглядит тип color для clrDarkOrange, BGR=(255,140,0)
0x00FF8C00 - представление ARGB с альфа-каналом=0x00, ARGB=(0,255,140,0)
RGB как color=(0,140,255) альфа-канал=85
0x55FF8C00 - представление ARGB с альфа-каналом=0x55, ARGB=(85,255,140,0)
0x55FF8C00 - результат ColorToARGB(clrDarkOrange,0x55)
*/

```

## Интерфейсы

Интерфейс предназначен для определения определённого функционала, который класс впоследствии может реализовывать. Фактически, это класс, который не может содержать члены и не может иметь конструктор и/или деструктор. Все объявленные в интерфейсе методы являются чисто виртуальными, даже без явного определения.

Определяется интерфейс с помощью ключевого слова `interface`, как показано в примере:

```

//--- базовый интерфейс для описания животных
interface IAnimal
{
//--- методы интерфейса по умолчанию имеют public-доступ
    void Sound(); // звук, который издает животное
};

//+-----+
//|  класс CCat наследуется от интерфейса IAnimal          |
//+-----+
class CCat : public IAnimal
{
public:
    CCat() { Print("Cat was born"); }
    ~CCat() { Print("Cat is dead"); }
//--- реализуем метод Sound интерфейса IAnimal
    void Sound(){ Print("meou"); }
};

//+-----+
//|  класс CDog наследуется от интерфейса IAnimal          |
//+-----+
class CDog : public IAnimal
{
public:

```

```

CDog() { Print("Dog was born"); }
~CDog() { Print("Dog is dead"); }

//--- реализуем метод Sound интерфейса IAnimal
void Sound() { Print("guaf"); }
};

//-----+
//| Script program start function |
//-----+

void OnStart()
{
//--- массив указателей на объекты типа IAnimal
IAnimal *animals[2];

//--- породим потомков IAnimal и сохраним указатели на них в массив
animals[0]=new CCat;
animals[1]=new CDog;

//--- вызовем метод Sound() базового интерфейса IAnimal для каждого потомка
for(int i=0;i<ArraySize(animals);++i)
    animals[i].Sound();

//--- удалим объекты
for(int i=0;i<ArraySize(animals);++i)
    delete animals[i];

//--- результат выполнения
/*
    Cat was born
    Dog was born
    meou
    guaf
    Cat is dead
    Dog is dead
*/
}
}

```

Как и в случае [абстрактных классов](#), нельзя создавать объект интерфейса без наследования. Интерфейс может наследоваться только от других интерфейсов и может выступать предком для класса. При этом интерфейс всегда имеет [публичную видимость](#).

Интерфейс нельзя объявить внутри объявления класса или структуры, но при этом указатель на интерфейс можно сохранить в переменную типа [void \\*](#). Вообще говоря, в переменную типа [void \\*](#) можно сохранить указатель на объект любого класса. Для того чтобы преобразовать указатель [void \\*](#) в указатель на объект конкретного класса, необходимо использовать оператор [dynamic\\_cast](#). В случае, когда преобразование невозможно, результатом операции [dynamic\\_cast](#) будет [NULL](#).

#### Смотри также

[Объектно-ориентированное программирование](#)

## Объект динамического массива

### Динамические массивы

Допускается объявление не более чем 4-мерного [массива](#). При объявлении динамического массива (массива с неуказанным значением в первой паре квадратных скобок) компилятор автоматически создает переменную указанной выше структуры (объект динамического массива) и обеспечивает код для правильной инициализации.

Динамические массивы автоматически освобождаются при выходе за пределы области видимости блока, в котором они объявлены.

**Пример:**

```
double matrix[] [10] [20]; // 3-мерный динамический массив
ArrayResize(matrix, 5); // задали размер первого измерения
```

### Статические массивы

При явном указании всех значимых размерностей массива компилятор заранее распределяет необходимый размер памяти. Такой массив называется статическим. Тем не менее, компилятор дополнительно распределяет память под объект динамического массива, который (объект) связан с заранее распределенным статическим буфером (областью памяти для хранения массива).

Создание объекта динамического массива обусловлено возможной необходимостью передавать данный статический массив в качестве параметра в какую-либо функцию.

**Примеры:**

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

### Массивы в составе структур

При объявлении статического массива в качестве члена структуры объект динамического массива не создается. Это сделано для совместимости структур данных, используемых в Windows API.

Однако статические массивы, объявленные в качестве членов структур, также можно передавать в MQL5-функции. В этом случае при передаче параметра будет создан временный объект динамического массива, связанный со статическим массивом - членом структуры.

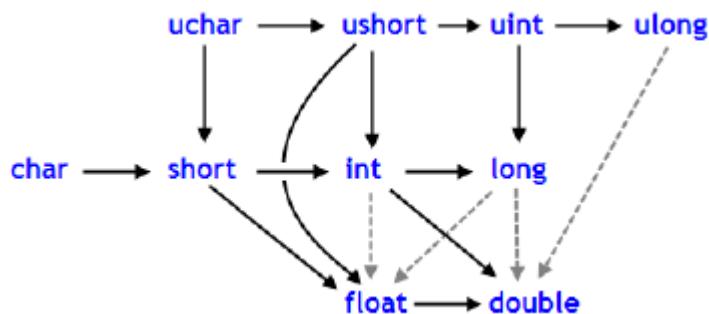
**Смотри также**

[Операции с массивами](#), [Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Приведение типов

### Преобразование числовых типов

Часто возникает необходимость преобразовать один числовой тип в другой. Не каждый числовой тип допустимо преобразовать в другой, допустимые преобразования в MQL5 показаны на схеме:



Сплошные линии со стрелками обозначают преобразования, которые выполняются без потери информации. Вместо типа `char` может выступать тип `bool` (оба занимают в памяти 1 байт), вместо типа `int` можно использовать тип `color` (по 4 байта), а вместо типа `long` допустим тип `datetime` (занимают по 8 байт). Четыре штриховые линии серого цвета, также со стрелками, означают преобразования, при которых может произойти потеря точности. Например, количество цифр в целом числе 123456789 (`int`) превышает количество цифр, которое может быть представлено типом `float`.

```

int n=123456789;
float f=n; // содержимое f равно 1.234567892E8
Print("n = ",n," f = ",f);
// результат n= 123456789 f= 123456792.00000
  
```

Число, преобразованное в тип `float`, имеет тот же порядок, но несколько меньшую точность. Преобразования, обратные черным стрелкам, осуществляется с возможной потерей информацией. Преобразования между `char` и `uchar`, `short` и `ushort`, `int` и `uint`, `long` и `ulong` (имеются ввиду преобразования в обе стороны), могут привести к потере информации.

В результате преобразования значения с плавающей точкой к целому типу дробная часть числа всегда отбрасывается. Если нужно округлить число с плавающей точкой до ближайшего целого числа (что во многих случаях является более полезным), необходимо использовать функцию [MathRound\(\)](#).

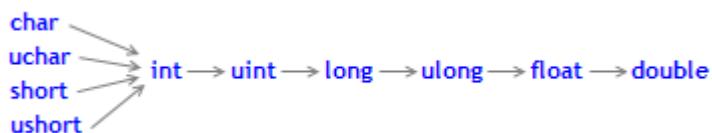
**Пример:**

```

//--- ускорение свободного падения
double g=9.8;
double round_g=(int)g;
double math_round_g=MathRound(g);
Print("round_g = ",round_g);
Print("math_round_g =",math_round_g);
/*
Результат:
round_g = 9
  
```

```
math_round_g = 10
*/
```

Если два значения объединяются бинарным оператором, то перед выполнением операции operand младшего типа преобразовывается к более старшему типу в соответствии с приоритетом, указанным на схеме:



Типы данных `char`, `uchar`, `short` и `ushort` в операциях безусловно приводятся к типу `int`.

### Примеры:

```
char c1=3;
//--- первый пример
double d2=c1/2+0.3;
Print("c1/2+0.3 = ",d2);
// Результат: c1/2+0.3 = 1.3

//--- второй пример
d2=c1/2.0+0.3;
Print("c1/2.0+0.3 = ",d2);
// Результат: c1/2.0+0.3 = 1.8
```

Вычисляемое выражение состоит из двух операций. В первом примере переменная `c1` типа `char` преобразуется ко временной переменной типа `int`, так как второй operand в операции деления, константа 2, имеет более старший тип `int`. В результате целочисленного деления 3/2 получается значение 1, которое имеет тип `int`.

Во второй операции первого примера вторым operandом выступает константа 0.3, которая имеет тип `double`, поэтому результат первой операции преобразуется во временную переменную типа `double` со значением 1.0.

Во втором примере переменная `c1` типа `char` преобразуется ко временной переменной типа `double`, так как второй operand в операции деления, константа 2.0, имеет тип `double`; дальнейших преобразований не производится.

## Приведение числовых типов

В выражениях языка MQL5 можно использовать как явное, так и неявное приведение типов. Явное преобразование типов записывается следующим образом:

```
var_1 = (тип) var_2;
```

В качестве переменной `var_2` может быть выражение или результат выполнения функции. Допускается также функциональная запись явного приведения типов:

```
var_1 = тип(var_2);
```

Рассмотрим явное преобразование на основании первого примера.

```
//--- третий пример
double d2=(double)c1/2+0.3;
Print("(double)c1/2+0.3 = ",d2);
// Результат: (double)c1/2+0.3 = 1.80000000
```

Перед выполнением операции деления переменная `c1` явно приводится к типу `double`. Теперь уже целочисленная константа 2 приводится к значению 2.0 типа `double`, так как в результате преобразования первый операнд получил тип `double`. Фактически, явное преобразование типов является одноместной операцией.

Кроме того, при попытке приведения типов результат может выйти за пределы допустимого диапазона. В этом случае произойдет усечение. Например:

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

Перед выполнением операций (кроме операций присваивания) происходит преобразование в тип, имеющий наибольший приоритет, а перед операциями присваивания - в целевой тип.

### Примеры:

```
int i=1/2;           // приведения типов нет, результат: 0
Print("i = 1/2 ",i);

int k=1/2.0;         // выражение приводится к типу double,
Print("k = 1/2 ",k); // затем приводится к целевому типу int, результат: 0

double d=1.0/2.0;    // приведения типов нет, результат: 0.5
Print("d = 1/2.0; ",d);

double e=1/2.0;      // выражение приводится к типу double,
Print("e = 1/2.0; ",e); // который совпадает с целевым типом, результат: 0.5

double x=1/2;        // выражение типа int приводится к целевому типу double,
Print("x = 1/2; ",x); // результат: 0.0
```

При преобразовании типа `long/ulong` в `double` может произойти потеря точности: если целое больше 9223372036854774784 или меньше -9223372036854774784.

```
void OnStart()
{
    long l_max=LONG_MAX;
    long l_min=LONG_MIN+1;
//--- найдем максимальное целое, которое не теряет точности при приведении к double
    while(l_max!=long((double)l_max))
        l_max--;
//--- найдем минимальное целое, которое не теряет точности при приведении к double
    while(l_min!=long((double)l_min))
```

```

l_min++;
//--- теперь выведем найденный интервал для целых чисел
PrintFormat("При приведении целого числа к double оно должно "
            "быть в интервале [%I64d, %I64d]", l_min, l_max);
//--- теперь посмотрим, что произойдет, если число выходит за этот интервал
PrintFormat("l_max+1=%I64d, double(l_max+1)=%.f, ulong(double(l_max+1))=%I64d",
            l_max+1,double(l_max+1),long(double(l_max+1)));
PrintFormat("l_min-1=%I64d, double(l_min-1)=%.f, ulong(double(l_min-1))=%I64d",
            l_min-1,double(l_min-1),long(double(l_min-1)));
//--- получим такой вывод
// При приведении целого числа к double оно должно быть в интервале [-9223372036854774785, 9223372036854774800]
// l_max+1=9223372036854774785, double(l_max+1)=9223372036854774800, ulong(double(l_max+1))=9223372036854774800
// l_min-1=-9223372036854774785, double(l_min-1)=-9223372036854774800, ulong(double(l_min-1))=-9223372036854774800
}

```

## Приведения для типа string

Тип `string` имеет самый высокий приоритет среди простых типов. Поэтому, если в операции один из операндов имеет тип `string`, то другой operand будет приведен к типу `string` автоматически. Следует иметь ввиду, что для типа `string` допустима единственная двуместная операция сложения. Допустимо явное приведение переменной типа `string` к любому числовому типу.

### Примеры:

```

string s1=1.0/8;           // выражение приводится к типу double,
Print("s1 = 1.0/8; ",s1);   // затем к целевому типу string,
// результат:"0.12500000" (строка, содержащая 10 символов)

string s2=NULL;            // деинициализация строки
Print("s2 = NULL; ",s2);    // результат: пустая строка
string s3="Ticket N"+12345; // выражение приводится к типу string
Print("s = \"Ticket N\"+12345 ",s3);

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

```

## Приведение типов указателей базовых классов к указателям производных классов

Объекты [открыто порожденного](#) класса могут также рассматриваться как объекты соответствующего ему базового класса. Это ведет к некоторым интересным следствиям. Например, вопреки тому факту, что объекты различных классов, порожденных одним базовым классом, могут существенно отличаться друг от друга, мы можем создать их связанный список (List), поскольку мы рассматриваем их как объекты базового типа. Но обратное неверно: объекты базового класса не являются автоматически объектами производного класса.

Можно использовать явное приведение типов для преобразования указателей базового класса в [указатели](#) производного класса. Но необходимо быть полностью уверенным в допустимости такого преобразования, так как в противном случае возникнет критическая ошибка времени выполнения и mq5-программа будет остановлена.

## Динамическое приведение типов с помощью оператора `dynamic_cast`

Существует возможность динамического приведения типов с помощью оператора `dynamic_cast`, который может быть применён только к указателям классов. При этом проверка корректности типов производится в момент выполнения программы. Это означает, что при использовании оператора `dynamic_cast` компилятор не производит проверку типа данных, используемого для приведения. В случае, если осуществляется преобразование указателя к типу данных, который не является фактическим типом объекта, результатом будет значение [NULL](#).

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

Параметр `type-id` в угловых скобках должен быть указателем на ранее определённый тип класса. Тип операнда `expression` (в отличии от C++) может быть любым, кроме [void](#).

**Пример:**

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

void OnStart()
{
    CBar bar;
//--- динамическое приведение типа указателя *bar к указателю *foo разрешено
    CFoo *foo = dynamic_cast<CFoo *>(&bar); // критической ошибки выполнения не возникнет
    Print(foo); // foo=NULL
//--- попытка явного приведения ссылки объекта типа Bar к объекту типа Foo запрещено
    foo=(CFoo *)&bar; // возникнет критическая ошибка выполнения
    Print(foo); // эта строка не будет выполнена
}
```

**Смотри также**

[Типы данных](#)

## Тип `void` и константа `NULL`

Синтаксически тип `void` является фундаментальным типом наравне с типами `char`, `uchar`, `bool`, `short`, `ushort`, `int`, `uint`, `color`, `long`, `ulong`, `datetime`, `float`, `double` и `string`. Этот тип используется либо для указания того, что функция не возвращает значения, либо в качестве параметра функции обозначает отсутствие параметров.

Предопределенная константная переменная `NULL` имеет тип `void`. Она может быть присвоена переменным любых других фундаментальных типов без преобразования. Также допускается сравнение переменных фундаментальных типов со значением `NULL`.

Пример:

```
//--- если строка не инициализирована, то присвоим ей наше предопределенное значение  
if(some_string==NULL) some_string="empty";
```

Также `NULL` можно сравнивать с указателями на объекты, созданные при помощи [оператора new](#).

Смотри также

[Переменные](#), [Функции](#)

## Пользовательские типы

Ключевое слово **typedef** в языке C++ позволяет создавать пользовательские типы данных - для этого достаточно определить новое имя типа данных для уже существующего типа данных. При этом сам новый тип данных не создается, а лишь определяется новое имя для уже существующего типа. Благодаря использованию пользовательских типов можно делать программы более гибкими: для этого иногда достаточно изменить **typedef**-инструкции с помощью макросов подстановки ([#define](#)). Использование пользовательских типов позволяет также улучшить читабельность кода, поскольку для стандартных типов данных с помощью **typedef** можно использовать собственные описательные имена. Общий формат записи инструкции для создания пользовательского типа:

```
typedef тип новое_имя;
```

Здесь элемент *тип* означает любой допустимый тип данных, а элемент *новое\_имя* - новое имя для этого типа. Важно отметить, что новое имя определяется только в качестве дополнения к существующему имени типа, а не для его замены. В языке MQL5 с помощью **typedef** можно создавать указатели на функции.

## Указатель на функцию

Указатель на функцию в общем виде определяются форматом записи

```
typedef тип_результата_функции (*Имя_типа_функции) (список_типов_входных параметров)
```

где после слова **typedef** задается сигнатура функции - количество и тип входных параметров, а также тип возвращаемого функцией результата. В качестве объяснения приведем простой пример создания и использования указателя на функцию:

```
//--- объявим указатель на функцию, которая принимает два параметра типа int
typedef int (*TFunc) (int,int);

//--- TFunc является типом и мы можем объявить переменную-указатель на функцию
TFunc func_ptr; // указатель на функцию

//--- объявим функции, которые соответствуют описанию TFunc
int sub(int x,int y) { return(x-y); } // вычитание одного числа из другого
int add(int x,int y) { return(x+y); } // сложение двух чисел
int neg(int x) { return(~x); } // инвертирование битов в переменной

//--- в переменную func_ptr можно сохранить адрес функции, чтобы в дальнейшем ее вызывать
func_ptr=sub;
Print(func_ptr(10,5));

func_ptr=add;
Print(func_ptr(10,5));

func_ptr=neg; // ошибка: neg не имеет тип int (int,int)
Print(func_ptr(10)); // ошибка: должно быть два параметра
```

В данном примере переменной *func\_ptr* можно присвоить функции *sub* и *add*, поскольку они имеют по два входных параметра типа [int](#), как это указано в определении указателя на функцию *TFunc*. А вот функция *neg* не может быть присвоена указателю *func\_ptr*, так как ее сигнатура отличается.

## Организации событийных моделей в пользовательском интерфейсе

С помощью указателей на функции удобно строить обработку событий при создании пользовательского интерфейса. Покажем на примере из раздела [CButton](#), как можно создавать кнопки и добавлять в них функции для обработки нажатия. Сначала определим указатель на функцию *TAction*, которая будет вызываться по нажатию кнопки, и создадим три функции в соответствии с описанием *TAction*.

```
//--- создадим пользовательский тип функции
typedef int (*TAction) (string,int);

//+-----+
//| Открывает файл
//+-----+
int Open(string name,int id)
{
    PrintFormat("Вызвана функция %s (name=%s id=%d)",__FUNCTION__,name,id);
    return(1);
}

//+-----+
//| Сохраняет файл
//+-----+
int Save(string name,int id)
{
    PrintFormat("Вызвана функция %s (name=%s id=%d)",__FUNCTION__,name,id);
    return(2);
}

//+-----+
//| Закрывает файл
//+-----+
int Close(string name,int id)
{
    PrintFormat("Вызвана функция %s (name=%s id=%d)",__FUNCTION__,name,id);
    return(3);
}
```

Затем произведем класс *MyButton* от [CButton](#), в котором добавим член *TAction*, являющийся указателем на функцию.

```
//+-----+
//| Создадим свой класс кнопки с функцией обработки событий
//+-----+
class MyButton: public CButton
{
private:
    TAction           m_action;          // обработчик событий графика
public:
    MyButton(void) {}
    ~MyButton(void) {}

    //--- конструктор с указанием текста кнопки и указателя на функцию для обработки событий
    MyButton(string text, TAction act)
    {
```

```

        Text(text);
        m_action=act;
    }

//--- установка собственной функции, которая будет вызываться из обработчика событий
void SetAction(TAction act){m_action=act;}

//--- стандартный обработчик событий графика
virtual bool OnEvent(const int id,const long &lparam,const double &dparam,const
{
    if(m_action!=NULL & lparam==Id())
    {
        //--- вызовем собственный обработчик m_action()
        m_action(sparam,(int)lparam);
        return(true);
    }
    else
        //--- вернем результат вызова обработчика из родительского класса CButton
        return(CButton::OnEvent(id,lparam,dparam,sparam));
}
};


```

Далее создадим производный класс `CControlsDialog` от `CAppDialog`, в котором добавим массив `m_buttons` для хранения кнопок типа `MyButton`, а также методы `AddButton(MyButton &button)` и `CreateButtons()`.

```

//+-----+
//| Класс CControlsDialog
//| Назначение: графическая панель для управления приложением
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CArrayObj             m_buttons;           // массив кнопок
public:
    CControlsDialog(void){};
    ~CControlsDialog(void){};

//--- create
virtual bool Create(const long chart,const string name,const int subwin,const
//--- добавление кнопки
bool AddButton(MyButton &button){return(m_buttons.Add(GetPointer(button)));}
protected:
    //--- создание кнопок
    bool CreateButtons(void);
};

//+-----+
//| Создание объекта CControlsDialog на графике
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))

```

```

        return(false);
    return(CreateButtons());
//---
}
//+-----+
//| defines
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)      // indent from left (with allowance)
#define INDENT_TOP           (11)      // indent from top (with allowance)
#define CONTROLS_GAP_X       (5)       // gap by X coordinate
#define CONTROLS_GAP_Y       (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)     // size by X coordinate
#define BUTTON_HEIGHT         (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)     // size by Y coordinate
//+-----+
//| Создание и добавление кнопок на панель CControlsDialog
//+-----+
bool CControlsDialog::CreateButtons(void)
{
//--- расчет координат кнопок
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2;
    int y2=y1+BUTTON_HEIGHT;
//--- добавим объекты кнопок вместе с указателями на функции
    AddButton(new MyButton("Open",Open));
    AddButton(new MyButton("Save",Save));
    AddButton(new MyButton("Close",Close));
//--- создадим кнопки графически
    for(int i=0;i<m_buttons.Total();i++)
    {
        MyButton *b=(MyButton*)m_buttons.At(i);
        x1=INDENT_LEFT+i*(BUTTON_WIDTH+CONTROLS_GAP_X);
        x2=x1+BUTTON_WIDTH;
        if(!b.Create(m_chart_id,m_name+"bt"+b.Text(),m_subwin,x1,y1,x2,y2))
        {
            PrintFormat("Failed to create button %s %d",b.Text(),i);
            return(false);
        }
//--- добавим каждую кнопку в контейнер CControlsDialog
        if(!Add(b))
            return(false);
    }
//--- succeed
    return(true);
}

```

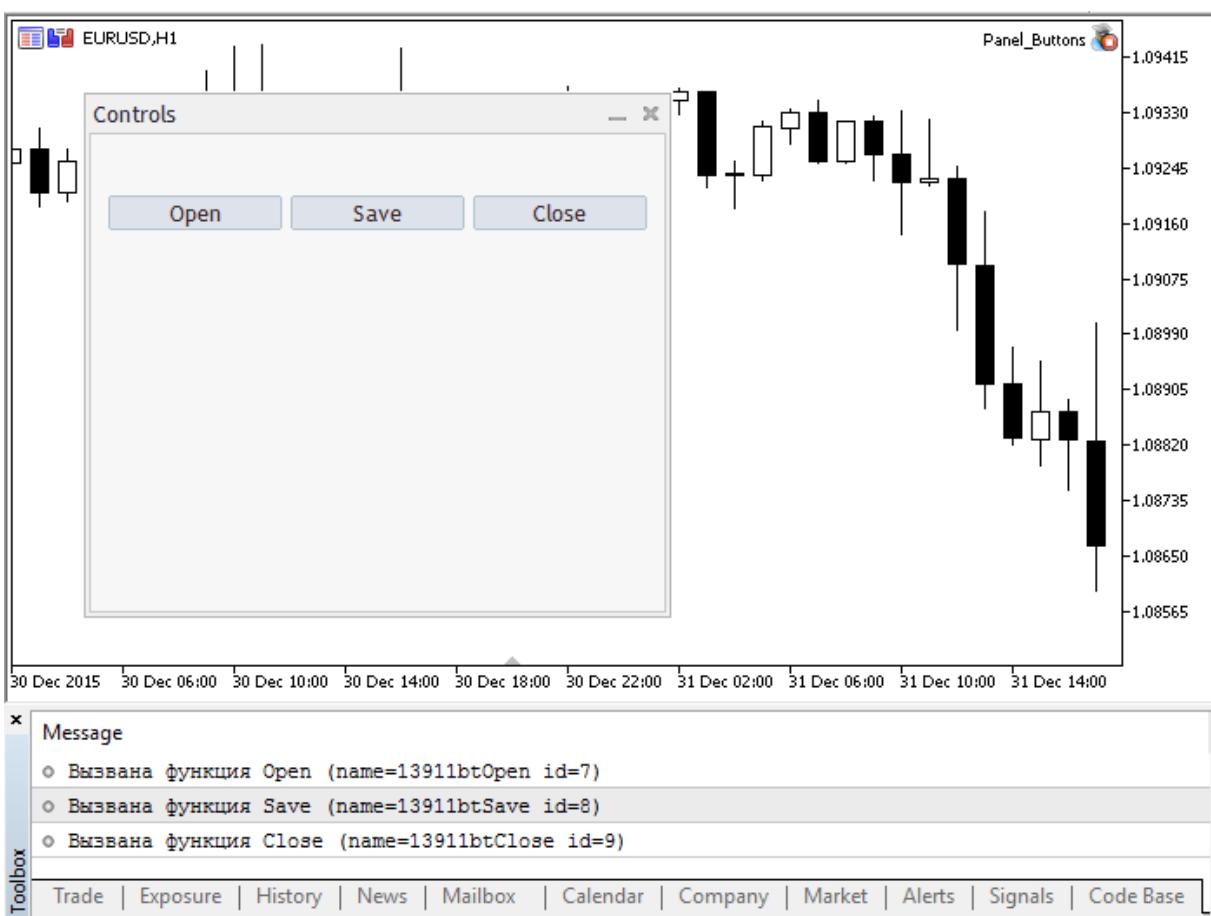
Теперь мы можем написать программу с использованием панели управления CControlsDialog, в которой создается 3 кнопки "Open", "Save" и "Close". При нажатии на кнопку вызывается соответствующая ей функция, которая прописана в виде указателя на функцию *TAction*.

```

//--- объявим объект на глобальном уровне, чтобы создать его автоматически при запуске
CControlsDialog MyDialog;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- теперь создадим объект на графике
    if (!MyDialog.Create(0, "Controls", 0, 40, 40, 380, 344))
        return (INIT_FAILED);
//--- запускаем приложение
    MyDialog.Run();
//--- успешная инициализация приложения
    return (INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- destroy dialog
    MyDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function
//+-----+
void OnChartEvent(const int id,           // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,   // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
//--- для событий графика вызываем обработчик из родительского класса (CAppDialog в данном
//     случае)
    MyDialog.ChartEvent(id, lparam, dparam, sparam);
}

```

Внешний вид запущенного приложения и результаты нажатия кнопок представлены на картинке.



### Полный исходный код программы

```

//+-----+
//|                                         Panel.Buttons.mq5 |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+



#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Панель с несколькими кнопками CButton"
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)        // indent from left (with allowar
#define INDENT_TOP           (11)        // indent from top (with allowar
#define CONTROLS_GAP_X       (5)         // gap by X coordinate
#define CONTROLS_GAP_Y       (5)         // gap by Y coordinate

```

```

//--- for buttons
#define BUTTON_WIDTH          (100)      // size by X coordinate
#define BUTTON_HEIGHT         (20)       // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT           (20)       // size by Y coordinate

//--- создадим пользовательский тип функции
typedef int (*TAction)(string,int);

//+-----+
//| Открывает файл
//+-----+
int Open(string name,int id)
{
    PrintFormat("Вызвана функция %s (name=%s id=%d)",__FUNCTION__,name,id);
    return(1);
}

//+-----+
//| Сохраняет файл
//+-----+
int Save(string name,int id)
{
    PrintFormat("Вызвана функция %s (name=%s id=%d)",__FUNCTION__,name,id);
    return(2);
}

//+-----+
//| Закрывает файл
//+-----+
int Close(string name,int id)
{
    PrintFormat("Вызвана функция %s (name=%s id=%d)",__FUNCTION__,name,id);
    return(3);
}

//+-----+
//| Создадим свой класс кнопки с функцией обработки событий
//+-----+
class MyButton: public CButton
{
private:
    TAction          m_action;           // обработчик событий графика
public:
    MyButton(void) {}
    ~MyButton(void) {}

//--- конструктор с указанием текста кнопки и указателя на функцию для обработки событий
    MyButton(string text,TAction act)
    {
        Text(text);
        m_action=act;
    }

//--- установка собственной функции, которая будет вызываться из обработчика события
}

```

```

void SetAction(TAction act) {m_action=act;}
//--- стандартный обработчик событий графика
virtual bool OnEvent(const int id,const long &lparam,const double &dparam,const
{
    if(m_action!=NULL & lparam==Id())
    {
        //--- вызовем собственный обработчик
        m_action(sparam,(int)lparam);
        return(true);
    }
    else
        //--- вернем результат вызова обработчика из родительского класса CButton
        return(CButton::OnEvent(id,lparam,dparam,sparam));
}
};

//+-----+
//| Класс CControlsDialog
//| Назначение: графическая панель для управления приложением
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CArrayObj m_buttons; // массив кнопок
public:
    CControlsDialog(void){};
    ~CControlsDialog(void){};

    //--- create
    virtual bool Create(const long chart,const string name,const int subwin,const
    //--- добавление кнопки
    bool AddButton(MyButton &button){return(m_buttons.Add(GetPointer(button)));}
protected:
    //--- создание кнопок
    bool CreateButtons(void);
};

//+-----+
//| Создание объекта CControlsDialog на графике
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
    return(CreateButtons());
};

//---+
//+-----+
//| Создание и добавление кнопок на панель CControlsDialog
//+-----+
bool CControlsDialog::CreateButtons(void)
{

```

```

//--- расчет координат кнопок
int x1=INDENT_LEFT;
int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
int x2;
int y2=y1+BUTTON_HEIGHT;
//--- добавим объекты кнопок вместе с указателями на функции
AddButton(new MyButton("Open",Open));
AddButton(new MyButton("Save",Save));
AddButton(new MyButton("Close",Close));
//--- создадим кнопки графически
for(int i=0;i<m_buttons.Total();i++)
{
    MyButton *b=(MyButton*)m_buttons.At(i);
    x1=INDENT_LEFT+i*(BUTTON_WIDTH+CONTROLS_GAP_X);
    x2=x1+BUTTON_WIDTH;
    if(!b.Create(m_chart_id,m_name+"bt"+b.Text(),m_subwin,x1,y1,x2,y2))
    {
        PrintFormat("Failed to create button %s %d",b.Text(),i);
        return(false);
    }
    //--- добавим каждую кнопку в контейнер CControlsDialog
    if(!Add(b))
        return(false);
}
//--- succeed
return(true);
}
//--- объявим объект на глобальном уровне, чтобы создать его автоматически при запуске
CControlsDialog MyDialog;
//-----+
//| Expert initialization function |+
//-----+
int OnInit()
{
//--- теперь создадим объект на графике
if(!MyDialog.Create(0,"Controls",0,40,40,380,344))
    return(INIT_FAILED);
//--- запускаем приложение
MyDialog.Run();
//--- успешная инициализация приложения
return(INIT_SUCCEEDED);
}
//-----+
//| Expert deinitialization function |+
//-----+
void OnDeinit(const int reason)
{
//--- destroy dialog
MyDialog.Destroy(reason);
}

```

```
    }

//+-----+
//| Expert chart event function
//+-----+

void OnChartEvent(const int id,           // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,   // event parameter of the double type
                  const string& sparam) // event parameter of the string type

{
//--- для событий графика вызываем обработчика из родительского класса (CAppDialog в
MyDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

#### Смотри также

[Переменные](#), [Функции](#)

## Указатели объектов

В MQL5 существует возможность динамически создавать объекты сложного типа. Это делается при помощи [оператора new](#), который возвращает описатель созданного объекта. Описатель имеет размер 8 байт. Синтаксически описатели объектов в MQL5 похожи на указатели в C++.

**Пример:**

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

В отличие от C++, переменная `hobject` из вышеприведенного примера не является указателем на память, а является дескриптором объекта. Кроме того, в языке MQL5 все объекты в параметрах функции обязательно должны передаваться по ссылке. Примеры передачи объектов в качестве параметра функции:

```
class Foo
{
public:
    string          m_name;
    int             m_id;
    static int      s_counter;
    //--- конструкторы и деструкторы
    Foo(void){Setup("noname");}
    Foo(string name){Setup(name);}
    ~Foo(void){};

    //--- инициализируем объект Foo
    void           Setup(string name)
    {
        m_name=name;
        s_counter++;
        m_id=s_counter;
    }
};

int Foo::s_counter=0;
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    //--- объявим объект как переменную с автоматическим созданием
    Foo fool;
    //--- вариант передачи объекта по ссылке
    PrintObject(fool);

    //--- объявим указатель на объект и создадим его с помощью оператора 'new'
    Foo *foo2=new Foo("foo2");
    //--- вариант передачи указателя на объект по ссылке
    PrintObject(foo2); // указатель на объект автоматически преобразуется компилятором

    //--- объявим массив объектов Foo
```

```

Foo foo_objects[5];
//--- вариант передачи массива объектов
PrintObjectsArray(foo_objects); // отдельная функция для передачи массива объектов

//--- объявим массив указателей на объекты типа Foo
Foo *foo_pointers[5];
for(int i=0;i<5;i++)
    foo_pointers[i]=new Foo("foo_pointer");
//--- вариант передачи массива указателей
PrintPointersArray(foo_pointers); // отдельная функция для передачи массива указате

//--- перед завершением работы обязательно удаляем объекты, созданные как указатели
delete(foo2);
//--- удаляем массив указателей
int size=ArraySize(foo_pointers);
for(int i=0;i<5;i++)
    delete(foo_pointers[i]);
//---

}

//-----+
//|   Объекты всегда передаются по ссылке |
//-----+
void PrintObject(Foo &object)
{
    Print(__FUNCTION__," : ",object.m_id," Object name=",object.m_name);
}

//-----+
//|   Передача массива объектов |
//-----+
void PrintObjectsArray(Foo &objects[])
{
    int size=ArraySize(objects);
    for(int i=0;i<size;i++)
        PrintObject(objects[i]);
}

//-----+
//|   Передача массива указателей на объект |
//-----+
void PrintPointersArray(Foo* &objects[])
{
    int size=ArraySize(objects);
    for(int i=0;i<size;i++)
        PrintObject(objects[i]);
}

```

#### Смотри также

[Переменные](#), [Инициализация переменных](#), [Область видимости и время жизни переменных](#),  
[Создание и уничтожение объектов](#)

## Ссылки. Модификатор & и ключевое слово this

### Передача параметров по ссылке

В MQL5 параметры [простых](#) типов можно передавать как по значению, так и по ссылке, в то время как параметры [сложных](#) типов всегда передаются по ссылке. Для указания компилятору на необходимость передачи параметра по ссылке, перед именем параметра ставится знак амперсанда **&**.

Передача параметра по ссылке означает передачу адреса переменной, поэтому все изменения, произведенные над переданным по ссылке параметром, сразу же отобразятся и в исходной переменной. Используя передачу параметров по ссылке можно организовать возврат одновременно нескольких результатов из функции. Чтобы предотвратить изменение переданного по ссылке параметра, необходимо использовать модификатор [const](#).

Таким образом, если входной параметр функции является [массивом](#), объектом структуры или класса, то в заголовке функции после типа переменной и перед ее именем ставится символ '**&**'.

#### Пример

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

public:
    void setArray(double &array[]);
};

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

В вышеприведенном примере объявлен [класс](#) CDemoClass, который содержит [приватный](#) член - массив m\_array[] типа [double](#). Объявлена [функция](#) setArray(), в которую по ссылке передается массив array[]. Если заголовок функции написать без указания передачи по ссылке, т.е. убрать знак амперсанда, то при попытке компиляции такого кода будет выдано сообщение об ошибке.

Несмотря на то, что массив передается по ссылке, мы не можем произвести присвоение одного массива другому. Необходимо сделать поэлементное копирование содержимого массива-источника в массив-приемник. Наличие символа **&** для массивов и структур при передаче в качестве параметра функции является обязательным при описании функции.

### Ключевое слово this

Переменная типа класса (объект) может передаваться как по ссылке, так и по [указателю](#). Указатель как и ссылка служит для того чтобы получать доступ к объекту. После объявления указателя объекта необходимо применить к нему оператор `new` для его создания и инициализации.

Зарезервированное слово `this` предназначено для получения ссылки объекта на самого себя, доступной внутри методов класса или структуры. `this` всегда ссылается на объект, в методе которого используется, а выражение `GetPointer(this)` даёт указатель объекта, членом которого является функция, в которой осуществлен вызов функции `GetPointer()`. В MQL5 функции не могут возвращать объекты, но разрешено возвращать указатель объекта.

Таким образом, если необходимо, чтобы функция вернула объект, то мы можем вернуть указатель этого объекта в виде `GetPointer(this)`. Добавим в описание класса `CDemoClass` функцию `getDemoClass()`, которая возвращает указатель объекта этого класса.

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

public:
    void setArray(double &array[]);
    CDemoClass *getDemoClass();
};

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

//+-----+
// |  возвращает собственный указатель
//+-----+
CDemoClass *CDemoClass::getDemoClass(void)
{
    return(GetPointer(this));
}
```

Структуры не имеют указателей, к ним нельзя применять операторы `new` и `delete`, и нельзя использовать `GetPointer(this)`.

#### Смотри также

[Указатели объектов](#), [Создание и уничтожение объектов](#), [Область видимости и время жизни переменных](#)

## 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.

В данном разделе рассматриваются следующие темы:

- [Выражения](#)
- [Арифметические операции](#)
- [Операции присваивания](#)
- [Операции отношения](#)
- [Логические операции](#)
- [Побитовые операции](#)
- [Другие операции](#)
- [Приоритеты и порядок операций](#)

## Выражения

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

**Примеры:**

```
a++; b = 10;           // несколько выражений расположены на одной строчке
//--- одно выражение разбито на несколько строк
x = (y * z) /
    (w + 2) + 127;
```

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

**Смотри также**

[Приоритеты и порядок операций](#)

## Арифметические операции

К арифметическим относятся аддитивные и мультипликативные операции:

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

Операция инкремента и декремента применяются только к переменным, к константам не применяются. Префиксные инкремент (++i) и декремент (--k) применяются к переменной непосредственно перед использованием этой переменной в выражении.

Постфиксные инкремент (i++) и декремент (k--) применяются к переменной сразу после использования этой переменной в выражении.

### Важное замечание

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

Могут возникнуть вычислительные проблемы при переносе вышеуказанного выражения из одной среды программирования в другую (например, из Borland C++ в MQL5). В общем случае порядок вычислений зависит от реализации компилятора. На практике существуют два способа реализации постдекремента (постинкремента):

1. постдекремент (постинкремент) применяется к переменной после вычисления всего выражения;
2. постдекремент (постинкремента) применяется к переменной сразу по месту операции.

В MQL5 в данный момент реализован первый способ вычисления постдекремента (постинкремента). Но даже обладая этим знанием лучше не экспериментировать с использованием данной тонкости.

### Примеры:

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

### Смотри также

[Приоритеты и порядок операций](#)

## Операции 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 $x$	$y ^= x;$

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.

В выражении оператор присваивания может присутствовать много раз. В этом случае обработка выражения ведется справа налево:

`y=x=3;`

Сначала переменной  $x$  будет присвоено значение 3, затем переменной  $y$  будет присвоено значение переменной  $x$ , то есть тоже 3.

Смотри также

[Приоритеты и порядок операций](#)

## 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	<code>a == b;</code>
Истина, если a не равно b	<code>a != b;</code>
Истина, если a меньше b	<code>a &lt; b;</code>
Истина, если a больше b	<code>a &gt; b;</code>
Истина, если a меньше или равно b	<code>a &lt;= b;</code>
Истина, если a больше или равно b	<code>a &gt;= b;</code>

Нельзя сравнивать два [вещественных числа](#) на равенство друг другу. В большинстве случаев два вроде бы одинаковых числа могут оказаться неравными из-за разницы значения в 15-ом знаке после запятой. Для корректного сравнения двух вещественных чисел необходимо сравнивать нормализованную разницу этих чисел с нулевым значением.

**Пример:**

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

**Смотри также**

[Приоритеты и порядок операций](#)

## Логические операции

### Логическое отрицание НЕ(!)

Операнд операции логического отрицания НЕ(!) должен иметь арифметический тип. Результат равен ИСТИНА(1), если значение операнда есть ЛОЖЬ(0), и равен ЛОЖЬ(0), если operand не равен ЛОЖЬ(0).

```
if(!a) Print("не 'a'");
```

### Логическая операция ИЛИ (||)

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

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

### Логическая операция И (&&)

Логическая операция И (&&) значений x и у. Значением выражения является ИСТИНА(1), если значения x и у истинны (не нуль). В противном случае - ЛОЖЬ(0).

### Короткая оценка логических операций

К логическим выражениям применяется схема так называемой "короткой оценки", то есть, вычисление выражения прекращается в тот момент, когда можно точно оценить результат выражения.

```
//---------------------------------------------------------------------+
//| Script program start function                                |
//+-----+
void OnStart()
{
    //--- первый пример короткой оценки
    if(func_false() && func_true())
        Print("Операция &&: Это сообщение вы никогда не увидите");
    else
        Print("Операция &&: Результат первого выражения false, поэтому второе выражение");
    //--- второй пример короткой оценки
    if(!func_false() || !func_true())
        Print("Операция ||: Результат первого выражения true, поэтому второе выражение");
    else
        Print("Операция ||: Это сообщение вы никогда не увидите");
}
//---------------------------------------------------------------------+
//| функция всегда возвращает false                               |
//+-----+
bool func_false()
```

```
{  
    Print("Функция func_false()");  
    return(false);  
}  
//+-----+  
//| функция всегда возвращает true  
//+-----+  
bool func_true()  
{  
    Print("Функция func_true()");  
    return(true);  
}
```

#### Смотри также

[Приоритеты и порядок операций](#)

## Побитовые операции

### Дополнение до единицы

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;
```

**Пример:**

```
char a='a',b;
b=~a;
Print("a = ",a, " b = ",b);
// Результат будет такой:
// a= 97   b= -98
```

### Сдвиг вправо

Двоичное представление x сдвигается вправо на y разрядов. Если сдвигаемое значение имеет беззнаковый тип, то осуществляется логический сдвиг вправо, то есть, освобождающиеся слева разряды будут заполняться нулями.

Если же сдвигаемое значение имеет знаковый тип, то осуществляется арифметический сдвиг вправо, то есть освобождающиеся слева разряды будут заполняться значением знакового бита (если число положительное, то значение знакового бита равно 0, если число отрицательное, то значение знакового бита равно 1)

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

**Пример:**

```
char a='a',b='b';
Print("Before: a = ",a, " b = ",b);
//--- произведем сдвиг вправо
b=a>>1;
Print("After: a = ",a, " b = ",b);
// Результат будет такой:
// Before: a = 97   b = 98
// After: a = 97   b = 48
```

### Сдвиг влево

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

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

**Пример:**

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

```
//--- произведем сдвиг влево
b=a<<1;
Print("After: a = ",a, " b = ",b);
// Результат будет такой:
// Before: a = 97   b = 98
// After: a = 97   b = -62
```

Не рекомендуется производить сдвиг на большее или равное число битов, чем разрядность сдвигаемой переменной, так как результат такой операции не определен.

При выполнении операции сдвига над 32-х битными типами для вычисления сдвига учитываются только младшие 5 бит, а для операций сдвига над 64-х битными типами значения сдвига определяется только младшими 6 битами. Покажем это на примере:

```
int a=305;      // тип int имеет размер 4 байта и является 32-х битными числом
int b=a;        // сделаем копию
int shift=37;   // зададим переменную сдвига
shift++;        // увеличим значения на 1, shift=38
//--- задаем сдвиг на 38 разрядов, но на самом деле будет произведен сдвиг на 6 разрядов
a=a>>shift;    // 38 в двоичном представлении будет выглядеть как '100110', младшие 6 бит
b=b>>6;        // над переменной b произведен сдвиг на 6 разрядов
//--- убедимся, что переменные a и b содержат одинаковое значение после операции сдвига
Print("a = ", a, " b = ", b);
```

## Побитовая операция И

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);
```

**Пример:**

```
char a='a',b='b';
//--- операция И
char c=a&b;
Print("a = ",a," b = ",b);
Print("a & b = ",c);
// Результат будет такой:
// a = 97   b = 98
// a & b = 96
```

## Побитовая операция ИЛИ

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;
```

**Пример:**

```
char a='a',b='b';
//--- операция ИЛИ
```

```

char c=a|b;
Print("a = ",a," b = ",b);
Print("a | b = ",c);
// Результат будет такой:
// a = 97   b = 98
// a | b = 99

```

## Побитовая операция исключающее ИЛИ

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;
```

**Пример:**

```

char a='a', b='b';
//--- операция исключающее ИЛИ
char c=a^b;
Print("a = ",a," b = ",b);
Print("a ^ b = ",c);
// Результат будет такой:
// a = 97   b = 98
// a ^ b = 3

```

Побитовые операции выполняются только с [целыми числами](#).

**Смотри также**

[Приоритеты и порядок операций](#)

## 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 генерирует критическую ошибку, и выполнение программы будет остановлено.

### 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:**

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

### 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]);
```

## Операция точка ( . )

Для прямого [доступа к публичным членам](#) структур и классов используется операция точки. Синтаксис

```
Имя_переменной_типа_структурь.Имя_члена
```

Пример:

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

## Операция разрешения контекста ( :: )

Каждая функция в MQL5-программе имеет свой контекст исполнения. Например, системная функция [Print\(\)](#) выполняется в глобальном контексте. [Импортируемые](#) функции вызываются в контексте соответствующего импорта. Функции-методы [классов](#) имеют контекст соответствующего класса. Синтаксис операции разрешения контекста:

```
[Имя_контекста]::Имя_функции(параметры)
```

Если имя контекста отсутствует, то это явное указание на использование глобального контекста. В случае отсутствия операции разрешения контекста, функция ищется в ближайшем контексте. В случае отсутствия функции в локальном контексте, поиск производится в глобальном контексте.

Также операция разрешения контекста используется для [определения функции](#)-члена класса.

```
тип Имя_класса::Имя_функции(описание_параметров)
{
// тело функции
}
```

Если в программе используются или могут в будущем использоваться несколько одноименных функций из разных контекстов исполнения, то может возникнуть неоднозначность. Порядок приоритета вызова функций без явного указания контекста:

1. Методы класса. Если функция с заданным именем в классе не задана, то ищем на следующем уровне.
2. Функции MQL5. Если такой функции в языке нет, то ищем на следующем уровне.
3. Глобальные функции, определенные пользователем. Если такой функции нет, то ищем на следующем уровне.
4. Импортируемые функции. Если функция не найдена среди импортируемых, то компилятор выдает ошибку.

Для устранения неоднозначности вызова функций явно указывайте область видимости с помощью операции разрешения контекста.

### Пример:

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

class CCheckContext
{
    int         m_id;
public:
    CCheckContext() { m_id=1234; }
protected:
    int         GetLastError() { return(m_id); }
};

class CCheckContext2 : public CCheckContext
{
    int         m_id2;
public:
    CCheckContext2() { m_id2=5678; }
    void        Print();
protected:
    int         GetLastError() { return(m_id2); }
};

void CCheckContext2::Print()
{
    ::Print("Terminal GetLastError",::GetLastError());
    ::Print("kernel32 GetLastError",kernel32::GetLastError());
    ::Print("parent GetLastError",CCheckContext::GetLastError());
    ::Print("our GetLastError",GetLastError());
}

//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
//---
```

```
CCheckContext2 test;
test.Print();
}
//+-----+
```

## Операция взятия размера типа данных или размера объекта любого типа данных ( sizeof )

С помощью операции **sizeof** можно определить размер памяти которая соответствует идентификатору или типу. Операция sizeof имеет следующий формат:

**Пример:**

```
sizeof(выражение)
```

В качестве выражения может быть использован любой идентификатор, либо имя типа, заключенное в скобки. Отметим, что не может быть использовано имя типа void, а идентификатор не может относится к полю битов или быть именем функции.

Если в качестве выражения указано имя статического массива (то есть, задана первая размерность), то результатом является размер всего массива (т.е. произведение числа элементов на длину типа). Если в качестве выражения указано имя динамического массива (первая размерность не задана), то результатом будет размер объекта [динамического массива](#).

Когда sizeof применяются к имени типа структуры или класса или к идентификатору имеющему тип структуры или класса, то результатом является фактический размер структуры или класса.

**Пример:**

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

Вычисления размера происходит на этапе компиляции.

**Смотри также**

[Приоритеты и порядок операций](#)

## Приоритеты и порядок 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 operandов.

**Внимание:** Приоритет выполнения операций в языке MQL5 соответствует приоритету, принятому в языке C++, и отличается от приоритета, заданного в языке MQL4.

Операция	Описание	Порядок выполнения
() [] . . .	Вызов функции Выделение элемента массива Выделение элемента структуры	Слева направо
! ~ - ++ -- (тип) sizeof	Логическое отрицание Побитовое отрицание (complement) Изменение знака Увеличение на единицу (increment) Уменьшение на единицу (decrement) Преобразование типа Определение размера в байтах	Справа налево
*	Умножение	Слева направо
/	Деление	
%	Деление по модулю	
+	Сложение	Слева направо
-	Вычитание	
<<	Сдвиг влево	Слева направо
>>	Сдвиг вправо	
< <=	Меньше, чем Меньше или равно	Слева направо
> >=	Больше, чем Больше или равно	
==	Равно	Слева направо
!=	Не равно	
&	Побитовая операция И	Слева направо
^	Побитовая операция исключающее ИЛИ (eXclude OR)	Слева направо
	Побитовая операция ИЛИ	Слева направо
&&	Логическая операция И	Слева направо

	Логическая операция ИЛИ	Слева направо
?:	Условная операция	Справа налево
= *= /= %= += -= <<= >>= &= ^=  =	Присваивание Умножение с присваиванием Деление с присваиванием Деление по модулю с присваиванием Сложение с присваиванием Вычитание с присваиванием Сдвиг влево с присваиванием Сдвиг вправо с присваиванием Побитовое И с присваиванием Исключающее ИЛИ с присваиванием Побитовое ИЛИ с присваиванием	Справа налево
,	Запятая	Слева направо

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

## 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.

Оператор	Описание
<a href="#">Составной оператор {}</a>	Один или более операторов любого типа, заключенные в фигурные скобки { }
<a href="#">Оператор-выражение ()</a>	Любое выражение, заканчивающееся точкой с запятой ()
Оператор <a href="#">return</a>	Прекращает выполнение текущей функции и возвращает управление вызвавшей программе
Условный оператор <a href="#">if-else</a>	Используется при необходимости сделать выбор
Условный оператор <a href="#">?:</a>	Более простой аналог условного оператора if-else
Оператор выбора <a href="#">switch</a>	Передает управление оператору, который соответствует значению выражения
Оператор цикла <a href="#">while</a>	Выполняет оператор до тех пор, пока проверяемое выражение не станет ложным. Проверка выражения производится перед каждой итерацией
Оператор цикла <a href="#">for</a>	Выполняет оператор до тех пор, пока проверяемое выражение не станет ложным. Проверка выражения производится перед каждой итерацией
Оператор цикла <a href="#">do-while</a>	Выполняет оператор до тех пор, пока проверяемое выражение не станет ложным. Проверка условия окончания производится в конце, после каждого прохода цикла. Тело цикла всегда выполняется по крайней мере один раз.
Оператор <a href="#">break</a>	Прекращает выполнение ближайшего вложенного внешнего оператора switch, while, do-while или for
Оператор <a href="#">continue</a>	Передает управление в начало ближайшего внешнего оператора цикла while, do-while или for
Оператор <a href="#">new</a>	Создает объект соответствующего размера и возвращает описатель созданного объекта.

Оператор delete

Удаляет созданный оператором new объект

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!");
```

**Смотри также**

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## 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;
}
```

**Смотри также**

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Оператор-выражение

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

### Оператор присваивания:

Идентификатор = выражение;

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

В выражении оператор присваивания может присутствовать много раз. В этом случае обработка выражения ведется справа налево.

### 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.

Смотри также

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Оператор возврата `return`

Оператор `return` прекращает выполнение текущей [функции](#) и возвращает управление вызвавшей программе. Результат вычисления выражения возвращается вызываемой функции. Выражение может 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.

Можно возвращать [простые типы](#), [простые структуры](#), [указатели объектов](#). При помощи оператора `return` нельзя возвращать любые массивы, объекты классов, переменные типа сложных структур.

**Смотри также**

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Условный оператор if-else

Оператор IF - ELSE используется при необходимости сделать выбор. Формально синтаксис имеет вид:

```
if (выражение)
    оператор1
else
    оператор2
```

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.

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");
```

### Смотри также

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Условный оператор ?:

Общая форма тернарного оператора выглядит так:

```
выражение1 ? выражение2 : выражение3
```

В качестве первого операнда - "выражение1" - может быть использовано любое выражение, результатом которого является значение типа [bool](#). Если результат равен [true](#), то выполняется оператор, заданный вторым operandом, то есть, "выражение2".

Если же первый operand равен [false](#), то выполняется третий operand - "выражение3". Второй и третий operandы, то есть "выражение2" и "выражение3", должны возвращать значения одного типа и не должны иметь тип [void](#). Результатом выполнения условного оператора является результат "выражения2" либо результат "выражение3", в зависимости от результата "выражение1".

```
//--- пронормируем разность между ценами открытия и закрытия на дневной размах
double true_range = (High==Low) ? 0 : (Close-Open) / (High-Low);
```

Эта запись эквивалентна следующей

```
double true_range;
if(High==Low)
    true_range=0; // если High и Low равны
else
    true_range=(Close-Open) / (High-Low); // если размах ненулевой
```

## Ограничения по использованию оператора

Оператор на основании значения "выражение1" должен вернуть одно из двух значений - либо "выражение2", либо "выражение3". Существует ряд ограничений на эти выражения:

- Нельзя смешивать [пользовательский тип](#) с [простым типом](#) или [перечислением](#). Для [указателя](#) допустимо использовать [NULL](#).
- Если типы значений простые, то типом оператора будет максимальный тип (смотри [Приведение типов](#)).
- Если одно из значений имеет тип перечисление, а второе является числовым типом, то перечисление заменяется на int и действует второе правило.
- Если оба значения являются значениями перечислений, то их типы должны быть одинаковыми, и типом оператора будет перечисление.

Ограничения для пользовательских типов (классов или структур):

- типы должны быть одинаковыми или один должен [наследоваться](#) от другого.
- если типы не одинаковы (наследование), то потомок неявно приводится к родителю, то есть типом оператора будет тип родителя.
- нельзя смешивать объект и указатель - либо оба выражения являются объектами, либо [указателями](#). Для указателя допустимо использовать [NULL](#).

### Примечание

Будьте внимательны при использовании условного оператора в качестве аргумента [перегруженной функции](#), так как тип результата условного оператора определяется на момент компиляции программы. И этот тип [определяется](#) как больший тип из типов "выражение2" и "выражение3".

**Пример:**

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

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

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

    func(Expression1?Expression2:Expression3); // получим предупреждение компилятора
    func(!Expression1?Expression2:Expression3); // получим предупреждение компилятора
}

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

**Смотри также**

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## 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 `int` или `uint`.

```
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` не обязательно должен быть объявлен и не 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.

### Примеры:

```
//--- первый 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;
}

//--- второй пример
string res="";
int i=0;
switch(i)
{
```

```
case 1:  
    res=i;break;  
default:  
    res="default";break;  
case 2:  
    res=i;break;  
case 3:  
    res=i;break;  
}  
Print(res);  
/*  
Результат  
default  
*/
```

#### Смотри также

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Оператор цикла while

Оператор **while** состоит из проверяемого выражения и оператора, который должен быть выполнен:

```
while (выражение)
    оператор;
```

Если выражение истинно, то оператор выполняется до тех пор, пока выражение не станет ложным. Если выражение ложно, то управление передается следующему оператору. Значение выражения определяется до выполнения оператора. Следовательно, если выражение ложно с самого начала, то оператор вообще не выполняется.

### Примечание

Если в цикле предполагается обрабатывать большое количество итераций, то рекомендуется проверять факт принудительного завершения программы с помощью функции [IsStopped\(\)](#).

### Пример:

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

### Смотри также

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Оператор цикла `for`

Оператор `for` состоит из трех выражений и выполняемого оператора:

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

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.

Оператор `for` эквивалентен следующей последовательности операторов:

```
выражение1;
while (выражение2)
{
    оператор;
    выражение3;
};
```

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 ','.

### Примечание

Если в цикле предполагается обрабатывать большое количество итераций, то рекомендуется проверять факт принудительного завершения программы с помощью функции [IsStopped\(\)](#).

### Примеры:

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

Смотри также

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Оператор цикла do while

Циклы for и while производят проверку окончания в начале, а не в конце цикла. Третий оператор цикла do - while проверяет условие окончания в конце, после каждого прохода цикла. Тело цикла всегда выполняется по крайней мере один раз.

```
do
    оператор;
while (выражение);
```

Сначала выполняется оператор, затем вычисляется выражение. Если оно истинно, то оператор выполняется снова и т.д. Если выражение становится ложным, цикл заканчивается.

### Примечание

Если в цикле предполагается обрабатывать большое количество итераций, то рекомендуется проверять факт принудительного завершения программы с помощью функции [IsStopped\(\)](#).

### Пример:

```
/*--- вычисление последовательности чисел Фибоначчи
int counterFibonacci=15;
int i=0,first=0,second=1;
int currentFibonacciNumber;
do
{
    currentFibonacciNumber=first+second;
    Print("i = ",i," currentFibonacciNumber = ",currentFibonacciNumber);
    first=second;
    second=currentFibonacciNumber;
    i++; // без этого оператора получится бесконечный цикл!
}
while(i<counterFibonacci && !IsStopped());
```

### Смотри также

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Оператор завершения `break`

Оператор `break` прекращает выполнение ближайшего вложенного внешнего оператора [switch](#), [while](#), [do-while](#) или [for](#). Управление передается оператору, следующему за заканчивающимся. 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;
```

Смотри также

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Оператор продолжения `continue`

Оператор `continue` передает управление в начало ближайшего внешнего оператора цикла [while](#), [do-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);
}
```

Смотри также

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Оператор создания объекта new

Оператор `new` автоматически создает объект соответствующего размера, вызывает конструктор объекта и возвращает [описатель созданного объекта](#). В случае неудачи оператор возвращает нулевой описатель, который можно сравнивать с константой `NULL`.

Оператор `new` может быть применен только к объектам [класса](#), к структурам он не применим.

Оператор не применяется для создания массивов объектов. Для этого следует использовать функцию [ArrayResize\(\)](#).

**Пример:**

```
//+-----+
//| Создание фигуры
//+-----+
void CTetrisField::NewShape()
{
    m_ypos=HORZ_BORDER;
//--- случайным образом создаём одну из 7 возможных фигур
    int nshape=rand()%7;
    switch(nshape)
    {
        case 0: m_shape=new CTetrisShape1; break;
        case 1: m_shape=new CTetrisShape2; break;
        case 2: m_shape=new CTetrisShape3; break;
        case 3: m_shape=new CTetrisShape4; break;
        case 4: m_shape=new CTetrisShape5; break;
        case 5: m_shape=new CTetrisShape6; break;
        case 6: m_shape=new CTetrisShape7; break;
    }
//--- отрисовываем
    if(m_shape!=NULL)
    {
//--- начальные установки
        m_shape.SetRightBorder(WIDTH_IN_PIXELS+VERT_BORDER);
        m_shape.SetYPos(m_ypos);
        m_shape.SetXPos(VERT_BORDER+SHAPE_SIZE*8);
//--- отрисуем
        m_shape.Draw();
    }
//---
}
```

Следует отметить, что описатель объекта не является указателем на память.

Объект, созданный с помощью оператора `new`, должен быть явно уничтожен оператором [delete](#).

**Смотри также**

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Оператор уничтожения объекта `delete`

Оператор `delete` удаляет созданный оператором `new` объект, вызывает деструктор соответствующего класса и освобождает память, занимаемую объектом. В качестве операнда используется действительный описатель существующего объекта. После выполнения операции `delete` описатель объекта становится недействительным.

Пример:

```
//--- удаляем уложенную фигуру
delete m_shape;
m_shape=NULL;
//--- создаём новую фигуру
NewShape();
```

Смотри также

[Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Функции

Всякая задача может быть разбита на подзадачи, каждую из которых можно либо непосредственно представить в виде кода, либо разбить на еще более мелкие подзадачи. Данный метод называется *пошаговым уточнением*. Функции служат для записи программного кода этих непосредственно решаемых подзадач. Код, описывающий, что делает функция, называется *определением функции*:

```
заголовок_функции
{
    инструкции
}
```

Все, что находится перед первой фигурной скобкой, составляет заголовок определения функции, а то, что находится между фигурными скобками, является телом определения функции. Заголовок функции включает в себя описание типа возвращаемого значения, имени ([идентификатора](#)) и [формальных параметров](#). Количество параметров, передаваемых в функцию, ограничено и не может превышать 64.

Функция может вызываться из других частей программы столько раз, сколько необходимо. По сути, возвращаемый тип, идентификатор функции и типы параметров составляют *прототип функции*.

Прототип функции - это объявление функции, но не ее определение. Благодаря явному объявлению возвращаемого типа и списка типов аргументов, при обращении к функциям возможны строгая проверка типов и неявные преобразования типов. Особенно часто объявления функций используются в классах для улучшения читаемости кода.

Определение функции должно точно соответствовать ее объявлению. Каждая объявленная функция должна быть определена.

**Пример:**

```
double                  // тип возвращаемого значения
linfunc (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. Можно возвращать [простые типы](#), [простые структуры](#), [указатели объектов](#). При помощи оператора *return* нельзя возвращать любые массивы, объекты классов, переменные типа сложных структур.

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,      // объявлено значение по умолчанию 0.0001
             int n,                // значение по умолчанию не указано !
             bool b,               // значение по умолчанию не указано !
             string s="passed string")
{
}
```

**Смотри также**

[Перегрузка](#), [Виртуальные функции](#), [Полиморфизм](#)

## 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 не 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, //определены значения по умолчанию
             int level=10);
//...
somefunc(); // неправильный вызов. первый обязательный параметр
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.
```

**Смотри также**

[Перегрузка](#), [Виртуальные функции](#), [Полиморфизм](#)

## Передача параметров

Существует два метода, с помощью которых машинный язык может передавать аргумент подпрограмме (функции). Первый способ - передача параметра по значению. Этот метод копирует значение [аргумента](#) в формальный параметр функции. Поэтому любые изменения этого параметра внутри функции не имеют никакого влияния на соответствующий аргумент вызова.

```
//+-----+
//| передача параметров по значению
//+-----+
double FirstMethod(int i,int j)
{
    double res;
//---
    i*=2;
    j/=2;
    res=i+j;
//---
    return(res);
}
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//---
    int a=14,b=8;
    Print("a и b перед вызовом:",a," ",b);
    double d=FirstMethod(a,b);
    Print("a и b после вызова:",a," ",b);
}
//--- результат выполнения скрипта
//  а и б перед вызовом: 14 8
//  а и б после вызова: 14 8
```

Второй способ - передача аргумента по ссылке. В этом случае ссылка на параметр (а не его значение) передается параметру функции. Внутри функции она используется для того, чтобы обратиться к фактическому параметру, указанному в вызове. Это означает, что изменения параметра будут влиять на аргумент, использованный для вызова функции.

```
//+-----+
//| передача параметров по ссылке
//+-----+
double SecondMethod(int &i,int &j)
{
    double res;
//---
    i*=2;
    j/=2;
    res=i+j;
```

```

//---
    return(res);
}
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//---

    int a=14,b=8;
    Print("а и b перед вызовом:",a," ",b);
    double d=SecondMethod(a,b);
    Print("а и b после вызова:",a," ",b);
}

//+-----+
//--- результат выполнения скрипта
// а и b перед вызовом: 14 8
// а и b после вызова: 28 4

```

MQL5 использует оба метода, за одним исключением: массивы, переменные типа структур и объекты классов всегда передаются по ссылке. Для того чтобы исключить изменения фактических параметров (аргументов, переданных при вызове функции) необходимо использовать спецификатор доступа [const](#). При попытке изменить содержимое переменной, объявленной со спецификатором *const*, компилятор выдаст ошибку.

## Примечание

Необходимо помнить, что параметры передаются в функцию задом наперед, то есть сначала вычисляется и передается самый последний параметр, затем предпоследний и так далее. Последним по очереди вычисляется и передается параметр, стоящий первым после открывающей скобки.

### Пример:

```

void OnStart()
{
//---

    int a[]={0,1,2};
    int i=0;

    func(a[i],a[i++],"Первый вызов (i = "+string(i)+")");
    func(a[i++],a[i],"Второй вызов (i = "+string(i)+")");
// Результат:
// Первый вызов(i=0) : par1 = 1      par2 = 0
// Второй вызов(i=1) : par1 = 1      par2 = 1

}
//+-----+
//|
//+-----+
void func(int par1,int par2,string comment)

```

```
{  
    Print(comment,": par1 = ",par1,"      par2 = ",par2);  
}
```

При первом вызове в приведенном примере сначала переменная *i* участвует в конкатенации строк:

```
"Первый вызов (i = "+string(i)+"")"
```

При этом ее значение не меняется. Затем переменная *i* участвует в вычислении элемента массива *a[i++]*, то есть после взятия *i*-го элемента массива переменная *i* инкрементируется. И только после этого вычисляется первый параметр с измененным значением переменной *i*.

Во втором вызове при вычислении всех трех параметров используется одно и тоже значение *i*, измененное на первом вызове функции, и только после вычисления первого параметра переменная *i* опять изменяется.

#### Смотри также

[Область видимости и время жизни переменных](#), [Перегрузка](#), [Виртуальные функции](#),  
[Полиморфизм](#)

## Перегрузка функций

Обычно в названии функции стремятся отобразить ее основное назначение. Читабельные программы, как правило, содержат разнообразные и грамотно подобранные [идентификаторы](#). Иногда различные функции используются для одних и тех же целей. Например, рассмотрим функцию, которая вычисляет среднее значение массива чисел двойной точности, и такую же функцию, но оперирующую массивом целых чисел. И ту, и другую удобно назвать AverageFromArray:

```
//+-----+
//| вычисление среднего для массива типа double
//+-----+
double AverageFromArray(const double & array[],int size)
{
    if(size<=0)
        return 0.0;
    double sum=0.0;
    double aver;
//---
    for(int i=0;i<size;i++)
    {
        sum+=array[i];      // сложение для double
    }
    aver=sum/size;          // просто делим сумму на количество
//---
    Print("Вычисление среднего для массива типа double");
    return aver;
}
//+-----+
//| вычисление среднего для массива типа int
//+-----+
double AverageFromArray(const int & array[],int size)
{
    if(size<=0)  return 0.0;
    double aver=0.0;
    int sum=0;
//---
    for(int i=0;i<size;i++)
        sum+=array[i];      // сложение для int
    aver=(double)sum/size;// приведем сумму к типу double и разделим
//---
    Print("Вычисление среднего для массива типа int");
    return aver;
}
```

Каждая функция содержит вывод сообщения посредством функции [Print\(\)](#):

```
Print("Вычисление среднего для массива типа int");
```

Компилятор выбирает нужную функцию в соответствии с типами аргументов и их количеством. Правило, по которому осуществляется этот выбор, называется *алгоритмом соответствия сигнатуре*. Под сигнатурой понимается список типов, который используется в объявлении функции.

**Пример:**

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//---
int a[5]={1,2,3,4,5};
double b[5]={1.1,2.2,3.3,4.4,5.5};
double int_aver=AverageFromArray(a,5);
double double_aver=AverageFromArray(b,5);
Print("int_aver = ",int_aver," double_aver = ",double_aver);
}
//--- Результат работы скрипта
// Вычисление среднего для массива типа int
// Вычисление среднего для массива типа double
// int_aver= 3.00000000    double_aver= 3.30000000
```

Перегрузка функций - это создание нескольких функций с одним именем, но с разными параметрами. Это означает, что в перегружаемых вариантах функции разным должно быть количество аргументов и/или их тип. Выбор конкретного варианта функции зависит от типов аргументов, полученных функцией. Конкретная функция выбирается в зависимости от соответствия списка аргументов при вызове функции списку параметров в объявлении функции.

Когда вызывается перегруженная функция, компилятор должен иметь алгоритм для выбора надлежащей функции. Алгоритм, который выполняет этот выбор, зависит от того, [преобразования](#) какого типа присутствуют. Наилучшее соответствие должно быть уникальным. Найденная функция должна быть наилучшим выбором среди остальных вариантов хотя бы по одному аргументу, и, в то же время, по остальным аргументам она должна подходить не хуже остальных.

Ниже приведен алгоритм соответствия для каждого аргумента.

## Алгоритм выбора перегруженной функции

- Использовать строгое соответствие (если это возможно).
- Попробовать стандартное повышение типа.
- Попробовать стандартное преобразование типа.

Стандартное повышение типа лучше, чем остальные стандартные преобразования. Повышение - это преобразование `float` в `double`, а также `bool`, `char`, `short` или `enum` в `int`. Кроме того, к стандартным преобразованиям относятся преобразования массивов похожих [целых типов](#). Похожими типами являются: `bool`, `char`, `uchar`, так как все три типа являются однобайтовыми целыми; двубайтовые целые `short` и `ushort`; 4-байтовые целые `int`, `uint` и `color`; `long`, `ulong` и `datetime`.

Несомненно, строгое соответствие является наилучшим. Для достижения такого соответствия могут использоваться [приведения](#). Компилятор не справится с двусмысленной ситуацией. Поэтому не следует полагаться на тонкие различия в типах и неявные преобразования, которые делают перегруженную функцию неясной.

Если вы сомневаетесь, используйте явные преобразования для обеспечения строгого соответствия.

Примеры перегруженных функций в MQL5 вы можете увидеть на примере функций [ArrayInitialize\(\)](#).

Правила перегрузки функций применимы к [перегрузке методов классов](#).

Перегрузка системных функций допускается, но при этом следует следить за тем, чтобы компилятор точно мог выбрать нужную функцию. Для примера, мы можем перегрузить системную функцию [MathMax\(\)](#) 4-мя различными способами, но только два варианта будут корректными.

**Пример:**

```
// 1. перегрузка допустима - функция отличается от встроенной в язык MathMax() по количеству параметров
double MathMax(double a,double b,double c);

// 2. перегрузка недопустима!
// количество параметров разное, но последний имеет значение по умолчанию
// это приводит к скрытию системной функции при вызове, что недопустимо
double MathMax(double a,double b,double c=DBL_MIN);

// 3. перегрузка допустима - нормальная перегрузка по типу параметров a и b
int MathMax(int a,int b);

// 4. перегрузка недопустима!
// количество и тип параметров не отличается от исходной double MathMax(double a,double b);
int MathMax(double a,double b);
```

**Смотри также**

[Перегрузка](#), [Виртуальные функции](#), [Полиморфизм](#)

## Перегрузка операций

Для удобства чтения и написания кода разрешается перегрузка некоторых операций. Оператор перегрузки записывается с помощью ключевого слова **operator**. Разрешена перегрузка следующих операций:

- бинарные `+, -, *, %, <<, >>, ==, !=, <, >, <=, >=, =, +=, -=, /=, *=, %=, &=, |=, ^=, <<=, >>=, &&, ||, &, |, ^;`
- унарные `+, -, ++, --, !, ~;`
- оператор присваивания `=;`
- оператор индексации `[]`.

Перегрузка операций позволяет использовать операционную нотацию (запись в виде простых выражений) к сложным объектам - структурам и классам. Запись выражений с использованием перегруженных операций упрощает восприятие исходного кода, так как более сложная реализация скрыта.

Для примера рассмотрим широко применяемые в математике комплексные числа, которые состоят из действительной и мнимой части. В языке MQL5 нет типа данных для представления комплексных чисел, но есть возможность создать новый тип данных в виде [структур или класса](#). Объявим структуру `complex` и определим в ней четыре метода, реализующие четыре арифметические операции:

```
//+-----+
//| Структура для операций с комплексными числами |
//+-----+
struct complex
{
    double re; // действительная часть
    double im; // мнимая часть
    //--- конструкторы
    complex():re(0.0),im(0.0) { }
    complex(const double r):re(r),im(0.0) { }
    complex(const double r,const double i):re(r),im(i) { }
    complex(const complex &o):re(o.re),im(o.im) { }
    //--- арифметические операции
    complex Add(const complex &l,const complex &r) const; // сложение
    complex Sub(const complex &l,const complex &r) const; // вычитание
    complex Mul(const complex &l,const complex &r) const; // умножение
    complex Div(const complex &l,const complex &r) const; // деление
};
```

Теперь мы можем объявлять в своем коде переменные, представляющие комплексные числа, и работать с ними.

Например так:

```
void OnStart()
{
    //--- объявим и инициализируем переменные комплексного типа
    complex a(2,4),b(-4,-2);
    PrintFormat("a=% .2f+i*% .2f, b=% .2f+i*% .2f",a.re,a.im,b.re,b.im);
```

```

//--- сложим два числа
complex z;
z=a.Add(a,b);
PrintFormat("a+b=% .2f+i*% .2f",z.re,z.im);
//--- умножим два числа
z=a.Mul(a,b);
PrintFormat("a*b=% .2f+i*% .2f",z.re,z.im);
//--- разделим два числа
z=a.Div(a,b);
PrintFormat("a/b=% .2f+i*% .2f",z.re,z.im);
//---
}

```

Но было бы удобнее для привычных арифметических операций с комплексными числами использовать привычные операторы "+", "-", "\*" и "/".

Ключевое слово **operator** используется для того чтобы определить функцию-член, осуществляющую преобразование типа. Унарные и бинарные операции для переменных-объектов класса могут быть перегружены как нестатические функции-члены. Они неявно действуют на объект класса.

Большинство бинарных операций можно перегружать как обычные функции, принимающие один или оба аргумента в виде переменной класса или в виде указателя на объект данного класса. Для нашего типа **complex** перегрузка в объявлении будет выглядеть так:

```

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

```

Полный пример скрипта:

```

//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
//--- объявим и инициализируем переменные комплексного типа
complex a(2,4),b(-4,-2);
PrintFormat("a=% .2f+i*% .2f,    b=% .2f+i*% .2f",a.re,a.im,b.re,b.im);
//a.re=5;
//a.im=1;
//b.re=-1;
//b.im=-5;
//--- сложим два числа
complex z=a+b;
PrintFormat("a+b=% .2f+i*% .2f",z.re,z.im);
//--- умножим два числа
z=a*b;
PrintFormat("a*b=% .2f+i*% .2f",z.re,z.im);
//--- разделим два числа

```

```

z=a/b;
PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
}

//+-----+
//| Структура для операций с комплексными числами |
//+-----+

struct complex
{
    double          re; // действительная часть
    double          im; // мнимая часть
    //--- конструкторы
    complex():re(0.0),im(0.0) { }
    complex(const double r):re(r),im(0.0) { }
    complex(const double r,const double i):re(r),im(i) { }
    complex(const complex &o):re(o.re),im(o.im) { }

    //--- арифметические операции
    complex        Add(const complex &l,const complex &r) const; // сложение
    complex        Sub(const complex &l,const complex &r) const; // вычитание
    complex        Mul(const complex &l,const complex &r) const; // умножение
    complex        Div(const complex &l,const complex &r) const; // деление
    //--- бинарные операторы
    complex operator+(const complex &r) const { return(Add(this,r)); }
    complex operator-(const complex &r) const { return(Sub(this,r)); }
    complex operator*(const complex &r) const { return(Mul(this,r)); }
    complex operator/(const complex &r) const { return(Div(this,r)); }
};

//+-----+
//| Сложение |
//+-----+

complex complex::Add(const complex &l,const complex &r) const
{
    complex res;
    //---
    res.re=l.re+r.re;
    res.im=l.im+r.im;
    //--- результат
    return res;
}

//+-----+
//| Вычитание |
//+-----+

complex complex::Sub(const complex &l,const complex &r) const
{
    complex res;
    //---
    res.re=l.re-r.re;
    res.im=l.im-r.im;
    //--- результат
}

```

```

        return res;
    }

//+-----+
// | Умножение |
//+-----+

complex complex::Mul(const complex &l,const complex &r) const
{
    complex res;
//---

    res.re=l.re*r.re-l.im*r.im;
    res.im=l.re*r.im+l.im*r.re;
//--- результат
    return res;
}

//+-----+
// | Деление |
//+-----+

complex complex::Div(const complex &l,const complex &r) const
{
//--- пустое комплексное число
    complex res(EMPTY_VALUE,EMPTY_VALUE);
//--- проверка на ноль
    if(r.re==0 && r.im==0)
    {
        Print(__FUNCTION__+": number is zero");
        return(res);
    }
//--- вспомогательные переменные
    double e;
    double f;
//--- выбор варианта вычисления
    if(MathAbs(r.im)<MathAbs(r.re))
    {
        e = r.im/r.re;
        f = r.re+r.im*e;
        res.re=(l.re+l.im*e)/f;
        res.im=(l.im-l.re*e)/f;
    }
    else
    {
        e = r.re/r.im;
        f = r.im+r.re*e;
        res.re=(l.im+l.re*e)/f;
        res.im=(-l.re+l.im*e)/f;
    }
//--- результат
    return res;
}

```

Большинство унарных операций для классов можно перегружать как обычные функции, принимающие единственный аргумент-объект класса или указатель на него. Добавим перегрузку унарных операций "-" и "!".

```
//+-----+
//| Структура для операций с комплексными числами |
//+-----+
struct complex
{
    double re;           // действительная часть
    double im;          // мнимая часть
    ...
    //--- унарные операторы
    complex operator-() const; // унарный минус
    bool operator!() const; // отрицание
};

...
//+-----+
//| Перегрузка оператора "унарный минус" |
//+-----+
complex complex::operator-() const
{
    complex res;
    //---
    res.re=-re;
    res.im=-im;
    //--- результат
    return res;
}

//+-----+
//| Перегрузка оператора "логическое отрицание" |
//+-----+
bool complex::operator!() const
{
    //--- действительная и мнимая часть комплексного числа равны нулю?
    return (re!=0 && im!=0);
}
```

Теперь мы можем проверять значение комплексного числа на ноль и получать отрицательное значение:

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- объявим и инициализируем переменные комплексного типа
```

```

complex a(2,4),b(-4,-2);
PrintFormat("a=% .2f+i*% .2f,    b=% .2f+i*% .2f",a.re,a.im,b.re,b.im);
//--- разделим два числа
complex z=a/b;
PrintFormat("a/b=% .2f+i*% .2f",z.re,z.im);
//--- комплексное число по умолчанию равно нулю (в конструкторе по умолчанию re==0 и im==0)
complex zero;
Print("!zero=",!zero);
//--- присвоим отрицательное значение
zero=-z;
PrintFormat("z=% .2f+i*% .2f,    zero=% .2f+i*% .2f",z.re,z.im, zero.re,zero.im);
PrintFormat("-zero=% .2f+i*% .2f",-zero.re,-zero.im);
//--- еще раз проверим на равенство нулю
Print("!zero=",!zero);
//---
}

```

Обратите внимание, что нам не пришлось в данном случае перегружать операцию присваивания `"=`, так как [структуры простых типов](#) можно копировать друг в друга напрямую. Таким образом, теперь мы можем писать код для расчетов с участием комплексных чисел в привычной манере.

Перегрузка оператора индексирования позволяет получать значения массивов, заключенных в объект, более простым и привычным способом, и это также способствует лучшей читаемости и пониманию исходного кода программ. Например, нам необходимо обеспечить доступ к символу в строке по указанной позиции. Стока в языке MQL5 является отдельным типом [string](#), который не является массивом символов, но с помощью перегруженной операции индексации в созданном классе `CString` мы можем обеспечить простую и прозрачную работу:

```

//+-----+
//| Класс для доступа к символам в строке как в массиве символов |
//+-----+
class CString
{
    string          m_string;

public:
    CString(string str=NULL):m_string(str) { }
    ushort operator[] (int x) { return(StringGetCharacter(m_string,x)); }
};

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- массив для получения символов из строки
    int      x[]={ 19,4,18,19,27,14,15,4,17,0,19,14,17,27,26,28,27,5,14,
                  17,27,2,11,0,18,18,27,29,30,19,17,8,13,6 };
    CString str("abcdefghijklmnopqrstuvwxyz[ ]CS");
    string   res;
    //--- составим фразу, набрав символы из переменной str

```

```

for(int i=0,n=ArraySize(x);i<n;i++)
{
    res+=ShortToString(str[x[i]]);
}
//--- выводем результат
Print(res);
}

```

Другой пример перегрузки операции индексирования - работа с матрицами. Матрица представляет собою двумерный динамический массив, размеры массивов заранее неопределены. Поэтому нельзя объявить массив вида array[][] без указания размера второго измерения и затем передавать этот массив в качестве параметра. Выходом может служить специальный класс CMatrix, который содержит в себе массив объектов класса CRow.

```

//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- операции сложения и умножения матриц
CMatrix A(3),B(3),C();
//--- готовим массивы под строки
double a1[3]={1,2,3}, a2[3]={2,3,1}, a3[3]={3,1,2};
double b1[3]={3,2,1}, b2[3]={1,3,2}, b3[3]={2,1,3};
//--- заполняем матрицы
A[0]=a1; A[1]=a2; A[2]=a3;
B[0]=b1; B[1]=b2; B[2]=b3;
//--- выводем матрицы в журнал "Эксперты"
Print("---- элементы матрицы А");
Print(A.String());
Print("---- элементы матрицы В");
Print(B.String());

//--- сложение матриц
Print("---- сложение матриц А и В");
C=A+B;
//--- вывод форматированного строкового представления
Print(C.String());

//--- умножение матриц
Print("---- произведение матриц А и В");
C=A*B;
Print(C.String());

//--- а теперь покажем как получать значения в стиле динамических массивов matrix[i]
Print("Выводим значения матрицы С поэлементно");
//--- перебираем в цикле строки матрицы - объекты CRow
for(int i=0;i<3;i++)
{

```

```

        string com="| ";
        //--- формируем для значения строки из матрицы
        for(int j=0;j<3;j++)
        {
            //--- получим элемент матрицы по номерам строки и столбца
            double element=C[i][j];// [i] - доступ к CRow в массиве m_rows[] ,
                                    // [j] - перегруженный оператор индексации в CRow
            com=com+StringFormat("a(%d,%d)=%G ; ",i,j,element);
        }
        com+="|";
        //--- выводим значения строки
        Print(com);
    }
}

//-----+
// | Класс "Строка" |
//-----+
class CRow
{
private:
    double      m_array[];
public:
    //--- конструкторы и деструктор
    CRow(void)           { ArrayResize(m_array,0);      }
    CRow(const CRow &r) { this=r;                      }
    CRow(const double &array[]);
    ~CRow(void) { };

    //--- количество элементов в строке
    int         Size(void) const   { return(ArraySize(m_array)); }

    //--- возвращает строку со значениями
    string      String(void) const;

    //--- оператор индексации
    double      operator[](int i) const { return(m_array[i]); }

    //--- операторы присваивания
    void        operator=(const double &array[]); // массив
    void        operator=(const CRow & r);          // другой объект CRow
    double      operator*(const CRow &o);           // объект CRow для умножения
};

//-----+
// | Конструктор для инициализации строки массивом |
//-----+
void  CRow::CRow(const double &array[])
{
    int size=ArraySize(array);

    //--- если массив не пустой
    if(size>0)
    {
        ArrayResize(m_array,size);
        //--- заполним значениями
    }
}

```

```

        for(int i=0;i<size;i++)
            m_array[i]=array[i];
    }
//---
}
//+-----+
//| Операция присваивания для массива |
//+-----+
void CRow::operator=(const double &array[])
{
    int size=ArraySize(array);
    if(size==0)
        return;
//--- заполняем массив значениями
    ArrayResize(m_array,size);
    for(int i=0;i<size;i++)
        m_array[i]=array[i];
//---
}
//+-----+
//| Операция присваивания для CRow |
//+-----+
void CRow::operator=(const CRow &r)
{
    int size=r.Size();
    if(size==0)
        return;
//--- заполняем массив значениями
    ArrayResize(m_array,size);
    for(int i=0;i<size;i++)
        m_array[i]=r[i];
//---
}
//+-----+
//| Оператор умножения на другую строку |
//+-----+
double CRow::operator*(const CRow &o)
{
    double res=0;
//--- проверки
    int size=Size();
    if(size!=o.Size() || size==0)
    {
        Print(__FUNCSIG__,": Ошибка умножения двух матриц, не совпадают размеры");
        return(res);
    }
//--- умножим поэлементно массивы и сложим произведения
    for(int i=0;i<size;i++)
        res+=m_array[i]*o[i];
}

```

```

//--- результат
    return(res);
}
//+-----+
//| Возвращает форматированное строковое представление |
//+-----+
string CRow::String(void) const
{
    string out="";
//--- если размер массива больше нуля
    int size=ArraySize(m_array);
//--- работаем только при ненулевом кол-ве элементов в массиве
    if(size>0)
    {
        out="{";
        for(int i=0;i<size;i++)
        {
            //--- собираем значения в строку
            out+=StringFormat(" %G;",m_array[i]);
        }
        out+=" }";
    }
//--- результат
    return(out);
}
//+-----+
//| Класс "Матрица" |
//+-----+
class CMatrix
{
private:
    CRow m_rows[];
public:
    //--- конструкторы и деструктор
    CMatrix(void);
    CMatrix(int rows) { ArrayResize(m_rows,rows); }
    ~CMatrix(void){};

    //--- получение размеров матрицы
    int Rows() const { return(ArraySize(m_rows)); }
    int Cols() const { return(Rows()>0? m_rows[0].Size():0); }

    //--- возвращает значения столбца в виде строки CRow
    CRow GetColumnAsRow(const int col_index) const;

    //--- возвращает строку со значениями матрицы
    string String(void) const;

    //--- оператор индексации возвращает строку по ее номеру
    CRow *operator[](int i) const { return(GetPointer(m_rows[i])); }

    //--- оператор сложения
    CMatrix operator+(const CMatrix &m);
}

```

```

//--- оператор умножения
CMatrix operator*(const CMatrix &m);
//--- оператор присваивания
CMatrix *operator=(const CMatrix &m);
};

//+-----+
//| Конструктор по умолчанию, создает массив строк нулевого размера |
//+-----+
CMatrix::CMatrix(void)
{
//--- нулевое количество строк в матрице
ArrayResize(m_rows,0);
//---
}

//+-----+
//| Возвращает значения столбца в виде строки CRow |
//+-----+
CRow CMatrix::GetColumnAsRow(const int col_index) const
{
//--- переменная для получения значений из столбца
CRow row();
//--- количество строк в матрице
int rows=Rows();
//--- если количество строк больше нуля, выполняем операцию
if(rows>0)
{
//--- массив для получения значений столбца с индексом col_index
double array[];
ArrayResize(array,rows);
//--- заполнение массива
for(int i=0;i<rows;i++)
{
//--- проверка номера столбца для i-ой строки на выход за пределы массива
if(col_index>=this[i].Size())
{
Print(__FUNCSIG__,": Ошибка! Номер столбца ",col_index,>" размера строки ";
break; // row останется неинициализированным объектом
}
array[i]=this[i][col_index];
}
//--- создадим строку CRow на основе значений массива
row=array;
}
//--- результат
return(row);
}

//+-----+
//| Сложение двух матриц |
//+-----+

```

```

CMatrix CMATRIX::operator+(const CMATRIX &m)
{
//--- количество строк и столбцов в переданной матрице
    int cols=m.Cols();
    int rows=m.Rows();
//--- матрица для получения результата сложения
    CMATRIX res(rows);
//--- размеры матрицы должны совпадать
    if(cols!=Cols() || rows!=Rows())
    {
//--- нельзя произвести сложение
        Print(__FUNCSIG__,": Ошибка сложения двух матриц, не совпадают размеры");
        return(res);
    }
//--- вспомогательный массив
    double arr[];
    ArrayResize(arr,cols);
//--- перебираем строки для сложения
    for(int i=0;i<rows;i++)
    {
//--- запишем результаты сложений строк матриц в массив
        for(int k=0;k<cols;k++)
        {
            arr[k]=this[i][k]+m[i][k];
        }
//--- поместим массив в строку матрицы
        res[i]=arr;
    }
//--- вернем результат сложения матриц
    return(res);
}
//+-----+
// | Умножение двух матриц |
//+-----+
CMATRIX CMATRIX::operator*(const CMATRIX &m)
{
//--- кол-во столбцов первой матрицы кол-во строк в переданной матрице
    int cols1=Cols();
    int rows2=m.Rows();
    int rows1=Rows();
    int cols2=m.Cols();
//--- матрица для получения результата сложения
    CMATRIX res(rows1);
//--- матрицы должны быть согласованы
    if(cols1!=rows2)
    {
//--- нельзя произвести умножение
        Print(__FUNCSIG__,": Ошибка умножения двух матриц, формат не согласован "
              "- число столбцов в первом сомножителе должно быть равно числу строк во в"

```

```

        return(res);
    }
//--- вспомогательный массив
double arr[];
ArrayResize(arr,cols1);
//--- заполняем строки в матрице произведения
for(int i=0;i<rows1;i++) // перебираем строки
{
//--- обнулим массив-приемник
ArrayInitialize(arr,0);
//--- перебираем элементы в строке
for(int k=0;k<cols1;k++)
{
//--- возьмем из матрицы m значения k-го столбца в виде строки CRow
CRow column=m.GetColumnAsRow(k);
//--- перемножим две строки и запишем результат скалярного умножения векторов
arr[k]=this[i]*column;
}
//--- поместим массив arr[] в i-ую строку матрицы
res[i]=arr;
}
//--- вернем произведение двух матриц
return(res);
}

//-----+
//| Операция присваивания |
//-----+
CMATRIX *CMATRIX::operator=(const CMATRIX &m)
{
//--- узнаем и зададим количество строк
int rows=m.Rows();
ArrayResize(m_rows,rows);
//--- заполним наши строки значениями строк переданной матрицы
for(int i=0;i<rows;i++)
this[i]=m[i];
//---
return(GetPointer(this));
}

//-----+
//| Строковое представление матрицы |
//-----+
string CMATRIX::String(void) const
{
string out="";
int rows=Rows();
//--- формируем построчно
for(int i=0;i<rows;i++)
out=out+this[i].String()+"\r\n";
//--- результат
}

```

```
    return(out);
}
```

**Смотри также**

[Перегрузка](#), [Арифметические операции](#), [Перегрузка функций](#), [Приоритеты и порядок операций](#)

## Описание внешних функций

Внешние функции, определенные в другом модуле, должны быть явно описаны. Описание включает в себя тип возвращаемого значения, имя функции и набор входных параметров с их типами. 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      MessageBoxW(int hWnd ,string szText,string szCaption,int nType);
int      SendMessageW(int hWnd,int Msg,int wParam,int lParam);
#import "lib.ex5"
double   round(double value);
#import
```

С помощью импорта можно очень легко описывать функции, вызываемые из внешних DLL или скомпилированных EX5 библиотек. Библиотеками EX5 являются скомпилированные ex5-файлы, которые имеют свойство [library](#). Импортировать из EX5 библиотек можно только функции, описанные с [модификатором export](#).

При совместном использовании DLL и EX5-библиотек следует помнить о том, что библиотеки должны иметь разные имена (вне зависимости от каталогов их размещения). Все импортируемые функции получают область видимости "имя файла" библиотеки.

**Пример:**

```
#import "kernel32.dll"
int GetLastError();
#import "lib.ex5"
int GetLastError();
#import

class CFoo
{
public:
    int          GetLastError() { return(12345); }
    void         func()
    {
        Print(GetLastError());           // вызов метода класса
        Print(::GetLastError());        // вызов функции MQL5
        Print(kernel32::GetLastError()); // вызов функции kernel32.dll
        Print(lib::GetLastError());     // вызов функции lib.ex5
    }
};

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

{}

**Смотри также**

[Перегрузка](#), [Виртуальные функции](#), [Полиморфизм](#)

## Экспортирование функций

Существует возможность использования в mq5-программе функции, объявленной в другой mq5-программе с постмодификатором `export`. Такая функция называется экспортируемой, и она доступна для вызова из других программ после компиляции.

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

Данный модификатор указывает компилятору внести функцию в таблицу EX5-функций, экспортляемых данным исполняемым ex5-файлом. Только функции с таким модификатором становятся доступными ("видимыми") из других mq5-программ.

Свойство `library` указывает компилятору, что данный EX5-файл будет являться библиотекой, и компилятор проставит это в заголовке EX5.

Все функции, которые планируются как экспортляемые, должны быть помечены модификатором `export`.

**Смотри также**

[Перегрузка](#), [Виртуальные функции](#), [Полиморфизм](#)

## Функции обработки событий

В языке MQL5 предусмотрена обработка некоторых [предопределенных событий](#). Функции для обработки этих событий должны быть определены в программе MQL5: имя функции, тип возвращаемого значения, состав параметров (если они есть) и их типы должны строго соответствовать описанию функции-обработчика события.

Именно по типу возвращаемого значения и по типам параметров обработчик событий клиентского терминала идентифицирует функции, обрабатывающие то или иное событие. Если у соответствующей функции указаны иные, не соответствующие нижеследующим описаниям, параметры или указан иной тип возвращаемого значения, то такая функция не будет использоваться для обработки события.

### OnStart

Функция OnStart() является обработчиком события [Start](#), которое автоматически генерируется только для запущенных на выполнение скриптов. Должна иметь тип [void](#), параметров не имеет:

```
void OnStart();
```

Для функции OnStart() допустимо указывать тип возвращаемого значения [int](#).

### OnInit

Функция OnInit() является обработчиком события [Init](#). Может иметь тип [void](#) или [int](#), параметров не имеет:

```
void OnInit();
```

Событие [Init](#) генерируется сразу после загрузки эксперта или индикатора, для скриптов это событие не генерируется. Функция OnInit() используется для инициализации. Если OnInit() имеет возвращаемое значение типа [int](#), то ненулевой код возврата означает неудачную инициализацию и генерирует событие [Deinit](#) с кодом причины деинициализации [REASON\\_INITFAILED](#).

Для оптимизации входных параметров эксперта рекомендуется в качестве кода возврата использовать значения из перечисления [ENUM\\_INIT\\_RETCODE](#). Эти значения предназначены для организации управления ходом оптимизации, в том числе для выбора наиболее подходящих [агентов тестирования](#). Прямо при инициализации эксперта еще до запуска самого тестирования можно запросить информацию о конфигурации и ресурсах агента (количество ядер, объем свободной памяти и т.д.) с помощью функции [TerminalInfoInteger\(\)](#). И на основе полученной информации либо разрешить использовать данный агент тестирования, либо отказаться от него при оптимизации данного эксперта.

### ENUM\_INIT\_RETCODE

Идентификатор	Описание
INIT_SUCCEEDED	Инициализация прошла успешно, тестирование эксперта можно продолжать. Этот код означает то же самое, что и нулевое значение - инициализация эксперта в тестере прошла успешно.

INIT_FAILED	Неудачная инициализация, тестирование нет смысла продолжать из-за неустранимых ошибок. Например, не удалось создать индикатор, необходимый для работы эксперта. Возврат этого значения означает то же самое, что и возврат значения, отличного от нуля - инициализация эксперта в тестере прошла неудачно.
INIT_PARAMETERS_INCORRECT	Предназначен для обозначения программистом некорректного набора входных параметров, в общей таблице оптимизации строка результата с таким кодом возврата будет подсвечена красным цветом. Тестирование для данного набора параметров эксперта не будет выполняться, агент свободен для получения нового задания. При получении этого значения тестер стратегий гарантированно не будет передавать данное задание другим агентам для повторного выполнения.
INIT_AGENT_NOT_SUITABLE	Ошибок в работе программы при инициализации не возникло, но по каким-то причинам данный агент не подходит для проведения тестирования. Например, недостаточно оперативной памяти, нет <a href="#">поддержки OpenCL</a> и так далее. После возврата этого кода агент больше не будет получать заданий до самого конца <a href="#">данной оптимизации</a> .

Функция OnInit() типа void всегда означает удачную инициализацию.

## OnDeinit

Функция OnDeinit() вызывается при деинициализации и является обработчиком события [Deinit](#). Должна быть объявлена с типом `void` и иметь один параметр типа `const int`, который содержит [код причины деинициализации](#). Если объявлен иной тип, компилятор выдаст предупреждение, но функция вызываться не будет. Для скриптов событие Deinit не генерируется и поэтому использовать в скриптах функцию OnDeinit() нельзя.

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

Событие Deinit генерируется для экспертов и индикаторов в следующих случаях:

- перед переинициализацией в связи со сменой символа или периода графика, к которому прикреплена mq5-программа;
- перед переинициализацией в связи со сменой [входных параметров](#);
- перед выгрузкой mq5-программы.

## OnTick

Событие [NewTick](#) генерируется только для экспертов при поступлении нового тика по символу, к графику которого прикреплен эксперт. Функцию OnTick() бесполезно определять в пользовательском индикаторе или скрипте, поскольку событие NewTick для них не генерируется.

Событие Tick генерируется только для экспертов, но это не означает, что эксперты обязаны иметь функцию OnTick(), так как для экспертов генерируются не только события NewTick, но и события Timer, BookEvent и ChartEvent. Должна быть объявлена с типом [void](#), параметров не имеет:

```
void OnTick();
```

## OnTimer

Функция OnTimer() вызывается при наступлении события [Timer](#), которое генерируется системным таймером только для экспертов и индикаторов - использовать ее в скриптах нельзя. Периодичность наступления этого события устанавливается при подписке на получение функцией [EventSetTimer\(\)](#) уведомлений о событии Timer.

Отписывание от приема посылки событий таймера для конкретного эксперта производится функцией [EventKillTimer\(\)](#). Функция должна быть определена с типом void, параметров не имеет:

```
void OnTimer();
```

Рекомендуется вызывать функцию EventSetTimer() однократно в функции OnInit(), а функцию EventKillTimer() вызывать однократно в OnDeinit().

Каждый эксперт и каждый индикатор работает со своим таймером, и получает события только от него. При завершении работы mq5-программы таймер уничтожается принудительно, если он был создан, но не был отключен функцией [EventKillTimer\(\)](#).

## OnTrade

Функция вызывается при наступлении события [Trade](#), которое возникает при изменении списка [выставленных ордеров](#) и [открытых позиций](#), [истории ордеров](#) и [истории сделок](#). При любом торговом действии (выставлении отложенного ордера, открытии/закрытии позиции, установке стопов, срабатывании отложенных ордеров и т.п.) соответствующим образом изменяется история ордеров и сделок и/или список позиций и текущих ордеров.

```
void OnTrade();
```

Пользователь должен самостоятельно в коде реализовать проверку состояния торгового счета при получении этого события (если это необходимо по условиям торговой стратегии). Если вызов функции OrderSend() завершился успешно и вернул значение true - это означает, что торговый сервер поставил ордер в очередь на исполнение и присвоил ему номер тикета. Как только сервер обработает данный приказ, будет сгенерировано событие Trade. И если пользователь запомнил значение тикета, то при обработке события OnTrade() он может по этому тикету выяснить что именно случилось с ордером.

## OnTradeTransaction

В результате выполнения определенных действий с торговым счетом, его состояние изменяется. К таким действиям относятся:

- Отсылка торгового запроса любым MQL5-приложением в клиентском терминале при помощи функций [OrderSend](#) и [OrderSendAsync](#) и его последующее исполнение;
- Отсылка торгового запроса через графический интерфейс терминала и его последующее исполнение;
- Срабатывание отложенных ордеров и стоп-ордеров на сервере;
- Выполнение операций на стороне торгового сервера.

В результате данных действий, для счета выполняются торговые транзакции:

- обработка торгового запроса;
- изменение открытых ордеров;
- изменение истории ордеров;
- изменение истории сделок;
- изменение позиций.

Например, при отсылке рыночного ордера на покупку, он обрабатывается, для счета создается соответствующий ордер на покупку, происходит исполнение ордера, его удаление из списка открытых, добавление в историю ордеров, далее добавляется соответствующая сделка в историю и создается новая позиция. Все эти действия являются торговыми транзакциями. Приход каждой такой транзакции в терминал является событием [TradeTransaction](#). Оно вызывает обработчик [OnTradeTransaction\(\)](#):

```
void OnTradeTransaction(
    const MqlTradeTransaction& trans,           // структура торговой транзакции
    const MqlTradeRequest& request,          // структура запроса
    const MqlTradeResult& result            // структура ответа
);
```

Обработчик содержит три параметра:

- trans** - в данный параметр передается структура [MqlTradeTransaction](#), описывающая торговую транзакцию, примененную к торговому счету;
- request** - в данный параметр передается структура [MqlTradeRequest](#), описывающая торговый запрос;
- result** - в данный параметр передается структура [MqlTradeResult](#), описывающая результат исполнения торгового запроса.

Два последних параметра **request** и **result** заполняются значениями только для транзакции типа [TRADE\\_TRANSACTION\\_REQUEST](#), информацию о транзакции можно получить из параметра **type** переменной **trans**. Обратите внимание, что в этом случае поле **request\_id** в переменной **result** содержит идентификатор [торгового запроса](#) **request**, в результате выполнения которого и была произведена [торговая транзакция](#), описанная в переменной **trans**. Наличие идентификатора запроса позволяет связать выполненное действие (вызов функций [OrderSend](#) или [OrderSendAsync](#)) с результатом этого действия, передаваемым в [OnTradeTransaction\(\)](#).

Один торговый запрос, отправленный из терминала вручную или через торговые функции [OrderSend\(\)](#)/[OrderSendAsync\(\)](#), может порождать на торговом сервере несколько последовательных торговых транзакций. При этом очередность поступления этих транзакций в терминал не гарантирована, поэтому нельзя свой торговый алгоритм строить на ожидании поступления одних торговых транзакций после прихода других.

- Все типы торговых транзакций описываются в перечислении [ENUM\\_TRADE\\_TRANSACTION\\_TYPE](#).
- Структура MqlTradeTransaction, описывающая торговую транзакцию, заполняется по-разному в зависимости от типа транзакции. Например для транзакций типа TRADE\_TRANSACTION\_REQUEST необходимо анализировать только одно поле - type (тип торговой транзакции). Для получения дополнительной информации необходимо анализировать второй и третий параметры функции OnTradeTransaction (request и result). Более подробная информация приведена в разделе "[Структура торговой транзакции](#)".
- В описании торговой транзакции передается не вся доступная информация по ордерам, сделкам и позициям (например, комментарий). Для получения расширенной информации следует использовать функции [OrderGet\\*](#), [HistoryOrderGet\\*](#), [HistoryDealGet\\*](#) и [PositionGet\\*](#).

После применения торговых транзакций к клиентскому счету, они последовательно помещаются в очередь торговых транзакций терминала, откуда уже последовательно передаются в точку входа OnTradeTransaction в порядке поступления в терминал.

Во время обработки торговых транзакций экспертом при помощи обработчика OnTradeTransaction(), терминал продолжает обрабатывать вновь поступающие торговые транзакции. Таким образом, состояние торгового счета может измениться уже в процессе работы OnTradeTransaction(). Например, пока MQL5-программа обрабатывает событие добавления нового ордера, он может быть исполнен, удален из списка открытых и перемещен в историю. В дальнейшем программа будет уведомлена о всех этих событиях.

Длина очереди транзакций составляет 1024 элемента. В случае, если OnTradeTransaction() будет обрабатывать очередную транзакцию слишком долго, старые транзакции в очереди могут быть вытеснены более новыми.

- В общем случае нет точного соотношения по количеству вызовов OnTrade и OnTradeTransaction. Один вызов OnTrade соответствует одному или нескольким вызовам OnTradeTransaction.
- OnTrade вызывается после соответствующих вызовов OnTradeTransaction.

## OnTester

Функция OnTester() является обработчиком события [Tester](#), которое автоматически генерируется по окончании исторического тестирования эксперта на заданном интервале дат. Функция должна быть определена с типом double, параметров не имеет:

```
double OnTester();
```

Функция вызывается непосредственно перед вызовом функции OnDeinit() и имеет тип возвращаемого значения double. Функция OnTester() может быть использована только в экспертах при тестировании и предназначена в первую очередь для расчета некоторого значения, используемого в качестве критерия Custom max при генетической оптимизации входных параметров.

При генетической оптимизации сортировка результатов в пределах одного поколения производится по убыванию. То есть, лучшими с точки зрения критерия оптимизации считаются результаты с наибольшим значением (для критерия оптимизации Custom max в расчет принимаются значения, возвращенные функцией OnTester). Худшие значения при такой

сортовке помещаются в конец и впоследствии отбрасываются и не принимают участия в формировании следующего поколения.

## OnTesterInit

Функция OnTesterInit() является обработчиком события [TesterInit](#), которое автоматически генерируется перед началом оптимизации эксперта в тестере стратегий. Функция должна быть определена с типом void, параметров не имеет:

```
void OnTesterInit();
```

Эксперт, имеющий обработчик OnTesterDeinit() или OnTesterPass(), при запуске оптимизации автоматически загружается на отдельном графике терминала с указанными в тестере символом и периодом, и получает событие TesterInit. Функция предназначена для инициализации эксперта перед началом оптимизации для последующей [обработки результатов оптимизации](#).

## OnTesterPass

Функция OnTesterPass() является обработчиком события [TesterPass](#), которое автоматически генерируется при поступлении фрейма во время оптимизации эксперта в тестере стратегий. Функция должна быть определена с типом void, параметров не имеет:

```
void OnTesterPass();
```

Эксперт с обработчиком OnTesterPass() автоматически загружается на отдельном графике терминала с указанными для тестирования символом/периодом и получает во время оптимизации события TesterPass при получении фрейма. Функция предназначена для динамической обработки [результатов оптимизации](#) прямо "на лету", не дожидаясь её окончания. Добавление фреймов производится функцией [FrameAdd\(\)](#), которую можно вызывать по окончании одиночного прохода в обработчике [OnTester\(\)](#).

## OnTesterDeinit

Функция OnTesterDeinit() является обработчиком события [TesterDeinit](#), которое автоматически генерируется по окончании оптимизации эксперта в тестере стратегий. Функция должна быть определена с типом void, параметров не имеет:

```
void OnTesterDeinit();
```

Эксперт с обработчиком TesterDeinit() автоматически загружается на график при запуске оптимизации и получает событие TesterDeinit после её завершения. Функция предназначена для финальной обработки всех [результатов оптимизации](#).

## OnBookEvent

Функция OnBookEvent() является обработчиком события [BookEvent](#). Событие BookEvent генерируется для экспертов и индикаторов при изменении состояния стакана цен (Depth of Market). Должна иметь тип void и один параметр типа string:

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

Чтобы получать события BookEvent по любому символу, достаточно предварительно подписать на получение этих событий для этого символа с помощью функции [MarketBookAdd\(\)](#). Для того

чтобы отписаться от получения события BookEvent по конкретному символу, необходимо вызывать функцию [MarketBookRelease\(\)](#).

В отличие от других событий, событие BookEvent является широковещательным. Это означает, что достаточно одному эксперту подписаться на получение события BookEvent с помощью функции MarketBookAdd, все остальные эксперты, имеющие обработчик OnBookEvent(), будут получать это событие. Поэтому необходимо анализировать имя символа, которое передается в обработчик в качестве параметра `const string& symbol`.

## OnChartEvent

OnChartEvent() является обработчиком группы событий [ChartEvent](#):

- CHARTEVENT\_KEYDOWN – событие нажатия клавиатуры, когда окно графика находится в фокусе;
- CHARTEVENT\_MOUSE\_MOVE – события перемещения мыши и нажатия кнопок мыши (если для графика установлено свойство [CHART\\_EVENT\\_MOUSE\\_MOVE=true](#));
- CHARTEVENT\_OBJECT\_CREATE – событие создания графического объекта (если для графика установлено свойство [CHART\\_EVENT\\_OBJECT\\_CREATE=true](#));
- CHARTEVENT\_OBJECT\_CHANGE – событие изменения свойств объекта через диалог свойств;
- CHARTEVENT\_OBJECT\_DELETE – событие удаления графического объекта (если для графика установлено свойство [CHART\\_EVENT\\_OBJECT\\_DELETE=true](#));
- CHARTEVENT\_CLICK – событие щелчка мыши на графике;
- CHARTEVENT\_OBJECT\_CLICK – событие щелчка мыши на графическом объекте, принадлежащем графику;
- CHARTEVENT\_OBJECT\_DRAG – событие перемещения графического объекта при помощи мыши;
- CHARTEVENT\_OBJECT\_ENDEDIT – событие окончания редактирования текста в поле ввода графического объекта LabelEdit;
- CHARTEVENT\_CHART\_CHANGE – событие изменения графика;
- CHARTEVENT\_CUSTOM+n – идентификатор пользовательского события, где n находится в диапазоне от 0 до 65535.
- CHARTEVENT\_CUSTOM\_LAST – последний допустимый идентификатор пользовательского события (CHARTEVENT\_CUSTOM+65535).

Функция может вызываться в экспертах и индикаторах, должна иметь тип `void` и 4 параметра:

```
void OnChartEvent(const int id,           // идентификатор события
                  const long& lparam,    // параметр события типа long
                  const double& dparam,   // параметр события типа double
                  const string& sparam)  // параметр события типа string
{}
```

Для каждого типа события входные параметры функции OnChartEvent() имеют определенные значения, которые необходимы для обработки этого события. В таблице перечислены события и значения, которые передаются через параметры.

Событие	Значение параметра <code>id</code>	Значение параметра	Значение параметра	Значение параметра
---------	------------------------------------	--------------------	--------------------	--------------------

		lparam	dparam	sparam
Событие нажатия клавиатуры	CHARTEVENT_KEYDOWN	Код нажатой клавиши	Количество нажатий клавиши, сгенерированных за время её удержания нажатом состояния	Строковое значение битовой маски, описывающее статус кнопок клавиатуры
События мыши (если для графика установлено свойство <b>CHART_EVENT_MOUSE_MOVE=true</b> )	CHARTEVENT_MOUSE_MOVE	X координата	Y координата	Строковое значение битовой маски, описывающее статус кнопок мыши
Событие создания графического объекта (если для графика установлено свойство <b>CHART_EVENT_OBJECT_CREATE=true</b> )	CHARTEVENT_OBJECT_CREATE	—	—	Имя созданного графического объекта
Событие изменения свойств объекта через диалог свойств	CHARTEVENT_OBJECT_CHANGE	—	—	Имя измененного графического объекта
Событие удаления графического объекта (если для графика установлено свойство <b>CHART_EVENT_OBJECT_DELETE=true</b> )	CHARTEVENT_OBJECT_DELETE	—	—	Имя удаленного графического объекта
Событие щелчка мыши на графике	CHARTEVENT_CLICK	X координата	Y координата	—

Событие щелчка мыши на графическом объекте	CHARTEVENT_OBJECT_CLICK	X координата	Y координата	Имя графического объекта, на котором произошло событие
Событие перемещения графического объекта при помощи мыши	CHARTEVENT_OBJECT_DRAG	—	—	Имя перемещенного графического объекта
Событие окончания редактирования текста в поле ввода графического объекта "Поле ввода"	CHARTEVENT_OBJECT_ENDEDIT	—	—	Имя графического объекта "Поле ввода", в котором завершилось редактирование текста
Событие изменения графика	CHARTEVENT_CHART_CHANGE	—	—	—
Пользовательское событие с номером N	CHARTEVENT_CUSTOM+N	Значение, заданное функцией <a href="#">EventChartCustom()</a>	Значение, заданное функцией <a href="#">EventChartCustom()</a>	Значение, заданное функцией <a href="#">EventChartCustom()</a>

## OnCalculate

Функция OnCalculate() вызывается только в пользовательских индикаторах при необходимости произвести расчет значений индикатора по событию [Calculate](#). Обычно это происходит при поступлении нового тика по символу, для которого рассчитывается индикатор. При этом индикатор не обязательно должен быть прикреплен к какому-нибудь ценовому графику данного символа.

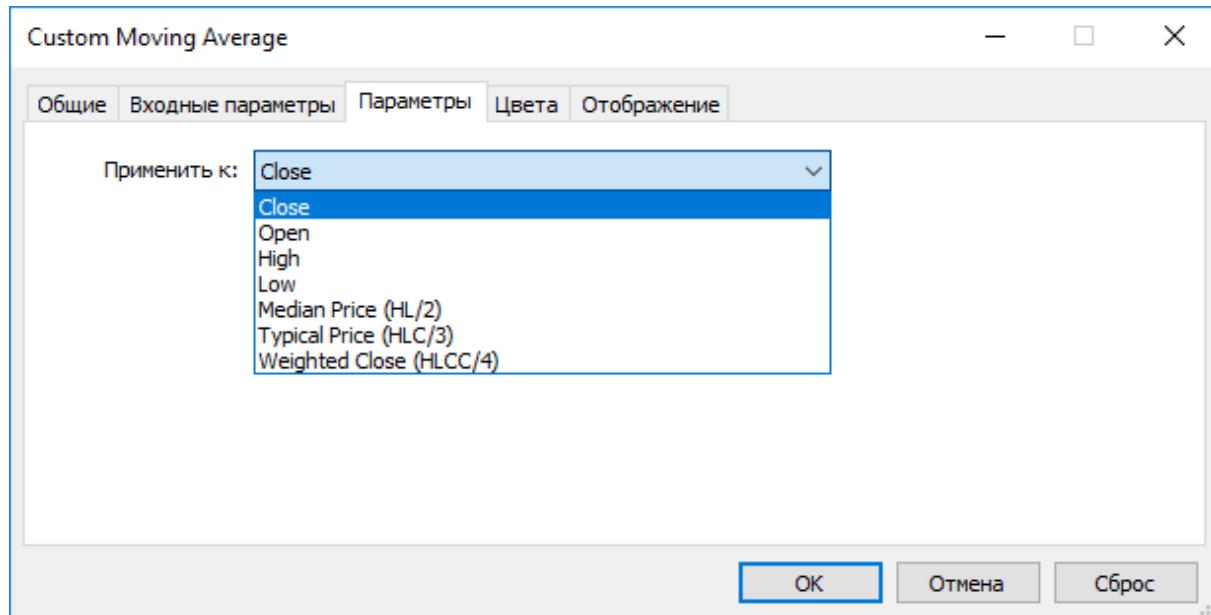
Функция OnCalculate() должна иметь тип возвращаемого значения int. Существует два варианта определения. В пределах одного индикатора нельзя использовать оба варианта функции.

Первая форма вызова предназначена для тех индикаторов, которые могут быть рассчитаны на одном буфере данных. Пример такого индикатора - Custom Moving Average.

```
int OnCalculate (const int rates_total,           // размер массива price[]
                const int prev_calculated, // обработано баров на предыдущем вызове
                const int begin,          // откуда начинаются значимые данные
                const double& price[])    // массив для расчета
{
```

В качестве массива price[] может быть передана одна из ценовых таймсерий либо рассчитанный буфер какого-либо индикатора. Чтобы определить направление индексации в массиве price[], необходимо вызывать функцию [ArrayGetAsSeries\(\)](#). Чтобы не зависеть от умолчаний, необходимо безусловно вызывать функцию [ArraySetAsSeries\(\)](#) для тех массивов, с которыми предполагается работать.

Выбор необходимой таймсерией или индикатора в качестве массива price[] осуществляется пользователем на вкладке "Parameters" при запуске индикатора. Для этого необходимо указать нужный элемент в выпадающем списке поля "Apply to".



Для получения значений пользовательского индикатора из других mq5-программ используется функция [iCustom\(\)](#), возвращающая хэндл индикатора для последующих операций. При этом также можно указать необходимый массив price[] или хэндл другого индикатора. Этот параметр должен передаваться последним в списке входных переменных пользовательского индикатора.

**Пример:**

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

В данном примере последним параметром передано значение PRICE\_TYPICAL (из перечисления [ENUM\\_APPLIED\\_PRICE](#)), которое указывает, что пользовательский индикатор будет построен по типическим ценам, полученным как (High+Low+Close)/3. Если параметр не указывается, то индикатор строится по значениям PRICE\_CLOSE, то есть по ценам закрытия каждого бара.

Другой пример, демонстрирующий передачу хендла индикатора последним параметром для указания массива price[], приведен в описании функции [iCustom\(\)](#).

Вторая форма вызова предназначена для всех остальных индикаторов, у которых для расчета используется более чем одна таймсерия.

```
int OnCalculate (const int rates_total,           // размер входных таймсерий
                const int prev_calculated,    // обработано баров на предыдущем вызове
                const datetime& time[],      // Time
                const double& open[],       // Open
                const double& high[],       // High
                const double& low[],        // Low
                const double& close[],      // Close
                const long& tick_volume[],   // Tick Volume
                const long& volume[],       // Real Volume
                const int& spread[])        // Spread
);
```

Параметры open[], high[], low[] и close[] содержат массивы с ценами открытия, максимальной, минимальной ценами и ценами закрытия текущего таймфрейма. Параметр time[] содержит массив со значениями времени открытия, параметр spread[] - массив, содержащий историю спредов (если спред предусмотрен для данного торгового инструмента). Параметры volume[] и tick\_volume[] содержат соответственно историю торгового и тикового объема.

Чтобы определить направление индексации в массивах time[], open[], high[], low[], close[], tick\_volume[], volume[] и spread[], необходимо вызывать функцию [ArrayGetAsSeries\(\)](#). Чтобы не зависеть от умолчаний, необходимо безусловно вызывать функцию [ArraySetAsSeries\(\)](#) для тех массивов, с которыми предполагается работать.

Первый параметр rates\_total содержит количество баров, доступных индикатору для расчета, и соответствует количеству баров, доступных на графике.

Необходимо отметить связь между значением, возвращаемым функцией OnCalculate() и вторым входным параметром prev\_calculated. Параметр prev\_calculated при вызове функции содержит значение, которое вернула функция OnCalculate() на предыдущем вызове. Это позволяет реализовать экономные алгоритмы расчета пользовательского индикатора с тем, чтобы избежать повторных расчетов для тех баров, которые не изменились с предыдущего запуска этой функции.

Для этого обычно достаточно вернуть значение параметра rates\_total, которое содержит количество баров при текущем вызове функции. Если с момента последнего вызова функции OnCalculate() ценовые данные были изменены (подкачана более глубокая история или были заполнены пропуски истории), то значение входного параметра prev\_calculated будет установлено в нулевое значение самим терминалом.

**Примечание:** если функция OnCalculate возвращает нулевое значение, то в окне DataWindow клиентского терминала значения индикатора не показываются.

Для лучшего понимания будет полезно запустить индикатор, код которого приложен ниже.

**Пример индикатора:**

```
#property indicator_chart_window
#property indicator_buffers 1
```

```

#property indicator_plots    1
//---- plot Line
#property indicator_label1  "Line"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDarkBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- indicator buffers
double          LineBuffer[];
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
    //--- indicator buffers mapping
    SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function                |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime& time[],
               const double& open[],
               const double& high[],
               const double& low[],
               const double& close[],
               const long& tick_volume[],
               const long& volume[],
               const int& spread[])
{
//--- получим количество доступных баров для текущих символов и периода на графике
    int bars=Bars(Symbol(),0);
    Print("Bars = ",bars,", rates_total = ",rates_total,", prev_calculated = ",prev_ca:
    Print("time[0] = ",time[0]," time[rates_total-1] = ",time[rates_total-1]);
//--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+

```

#### Смотри также

[Выполнение программ](#), [События клиентского терминала](#), [Работа с событиями](#)

## Переменные

### Объявление переменных

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.

Простыми типами являются:

- char, short, int, long, uchar, ushort, uint, ulong - целые числа;
- color - целое число, представляющее RGB-цвет;
- datetime - дата и время, беззнаковое целое число, содержащее количество секунд, прошедших с 0 часов 1 января 1970 года;
- bool - логические значения true и false;
- double - числа двойной точности с плавающей точкой;
- float - числа одинарной точности с плавающей точкой;
- string - символьные строки.

Примеры:

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

**Сложные или составные типы:**

Структуры - это составные типы данных, построенные с использованием других типов.

```
struct MyTime
{
    int hour;      // 0-23
    int minute;    // 0-59
    int second;    // 0-59
};

...
MyTime strTime; // переменная типа объявленной ранее структуры MyTime
```

До тех пор пока структура не объявлена, нельзя объявлять переменные типа структуры.

### 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.
```

```
MyTime t[100];           // массив содержащий элементы типа MyTime
```

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 t[6][49].

Статические массивы не могут быть представлены в виде таймсерий, то есть к ним не применима функция [ArraySetAsSeries\(\)](#), которая устанавливает доступ к элементам массива от конца массива к его началу. Если требуется обеспечить доступ к массиву как в [таймсерииях](#), используйте [объект динамического массива](#).

При доступе за пределы массива исполняющая подсистема генерирует критическую ошибку и выполнение программы будет остановлено.

## Спецификаторы доступа

Спецификаторы доступа указывают компилятору каким образом можно осуществлять доступ к переменным, членам структур или классов.

Спецификатор `const` объявляет переменную константой, и не позволяет изменять значение этой переменной в процессе выполнения программы. Допускается однократная инициализация переменной при ее объявлении.

### Пример

```
int OnCalculate (const int rates_total,          // размер массива price[]
                const int prev_calculated,    // обработано баров на предыдущем вызове
                const int begin,              // откуда начинаются значимые данные
                const double& price[])       // массив для расчета
{
```

Для доступа к членам структур и классов используются следующие спецификаторы:

- `public` - разрешает ничем неограниченный доступ к переменной или методу класса;
- `protected` - разрешает доступ со стороны методов данного класса, а также со стороны методов [публично наследуемых](#) классов. Иной доступ невозможен;
- `private` - разрешает доступ к переменным и методам класса только из методов данного класса.
- `virtual` - применим только к методам класса (но не к методам структур) и сообщает компилятору, что данный метод должен быть размещен в таблице виртуальных функций класса.

## Классы памяти

Существуют три класса памяти: `static`, `input` и `extern`. Эти модификаторы класса памяти явно указывают компилятору, что соответствующие переменные распределяются в предопределенной области памяти, называемой глобальным пулом. При этом данные модификаторы указывают на особую обработку данных переменных.

Если переменная, объявленная на локальном уровне, не является [статической](#), то распределение памяти под такую переменную производится автоматически на программном стеке. Освобождение памяти, выделенной под не статический массив, производится также автоматически при выходе за пределы области видимости блока, в котором массив объявлен.

#### Смотри также

[Типы данных](#), [Инкапсуляция и расширяемость типов](#), [Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#), [Статические члены класса](#)

## 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);
}
```

Область действия (или [область видимости](#)) переменной - это часть программы, в которой на переменную можно сослаться. Переменные, объявленные внутри блока (на внутреннем уровне), имеют областью действия [блок](#). Область действия блок начинается объявлением переменной и заканчивается конечной правой фигурной скобкой.

Локальные переменные, объявленные в начале функции, имеют область действия блок так же, как и [параметры функции](#), являющиеся локальными переменными. Любой блок может содержать объявления переменных. Если блоки вложены и [идентификатор](#) во внешнем блоке имеет такое же имя, как идентификатор во внутреннем блоке, идентификатор внешнего блока "невидим" (скрыт) до момента завершения работы внутреннего блока.

**Пример:**

```
void OnStart()
{
//---
    int i=5;      // локальная переменная функции
    {
        int i=10; // переменная функции
        Print("В блоке i = ",i); // результат i = 10;
    }
    Print("Вне блока i = ",i); // результат i = 5;
}
```

Это означает, что пока выполняется внутренний блок, он видит значения своих собственных локальных идентификаторов, а не значения идентификаторов с идентичными именами в охватывающем блоке.

**Пример:**

```
void OnStart()
{
//---
    int i=5;      // локальная переменная функции
    for(int i=0;i<3;i++)
```

```

Print("Внутри for i = ",i);
Print("Вне блока i = ",i);
}

/* Результат выполнения
Внутри for i = 0
Внутри for i = 1
Внутри for i = 2
Вне блока i = 5
*/

```

Локальные переменные, объявленные как [static](#), имеют областью действия блок, несмотря на то, что они существуют с самого начала выполнения программы.

## Стек

В каждой MQL5-программе под хранение локальных автоматически создаваемых переменных функций выделяется специальная область памяти, называемая стеком. Стек выделяется один на все функции и по умолчанию для индикаторов его размер равен 1Mb. В экспертах и скриптах размером стека можно управлять директивой компилятора [#property stacksize](#) (задает размер стека в байтах), для них по умолчанию под стек выделяется 8Mb.

[Статические](#) локальные переменные размещаются там же, где и другие статические и [глобальные](#) переменные, в специальной области памяти, существующей отдельно от стека. [Динамически](#) создаваемые переменные также используют отдельную от стека область памяти.

При каждом вызове функции для внутренних нестатических переменных отводится место на стеке. При выходе из функции память становится доступной для повторного использования.

Если из первой функции производится вызов второй функции, то та в свою очередь занимает на стеке необходимый объем под свои переменные из оставшейся стековой памяти. Таким образом при вложенных вызовах функций на стеке будет заниматься память последовательно под каждую функцию. Это может привести к нехватке памяти при очередном вызове функции, такая ситуация называется переполнением стека.

Поэтому для больших локальных данных лучше использовать динамическую память - при входе в функцию память под локальные нужды выделять в системе ([new](#), [ArrayResize\(\)](#)), а при выходе из функции производить освобождение памяти ([delete](#), [ArrayFree\(\)](#)).

### Смотри также

[Типы данных](#), [Инкапсуляция и расширяемость типов](#), [Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## 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. Если же формальный параметр объявлен с модификатором [const](#), то его значение не может быть изменено внутри функции.

**Пример:**

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

Формальные параметры могут быть [принициализированы](#) 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);
```

Параметры [простых типов](#) передаются по значению, то есть изменения соответствующей [локальной переменной](#) такого типа внутри вызываемой функции никак не отразятся в вызывающей функции. Массивы любых типов и данные типа структур всегда передаются по ссылке. Если необходимо запретить изменение массива или содержимого структуры, параметры этих типов должны быть объявлены с ключевым словом [const](#).

Существует возможность передавать по ссылке и параметры простых типов. В этом случае модификация таких параметров внутри вызываемой функции отразится на соответствующих переменных, переданных по ссылке. 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;
}
```

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

В функцию нельзя передать больше 64 параметров.

#### Смотри также

[Input переменные](#), [Типы данных](#), [Инкапсуляция и расширяемость типов](#), [Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Статические переменные

Класс памяти **static** определяет статическую переменную. Модификатор static указывается перед типом данных.

**Пример:**

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

Статическая переменная может быть [пронициализирована](#) соответствующей ее типу константой или константным выражением, в отличие от простой локальной переменной, которая может быть проинициализирована любым выражением.

Статические переменные существуют с момента выполнения программы и инициализируются однократно перед вызовом специализированной функции [OnInit\(\)](#). Если начальные значения не указаны, то переменные статического класса памяти принимают нулевые начальные значения.

[Локальные переменные](#), объявленные с ключевым словом static сохраняют свои значения в течение всего [времени существования](#) функции. При каждом следующем вызове функции такие локальные переменные содержат те значения, которые они имели при предыдущем вызове.

Любые переменные в блоке, кроме [формальных параметров](#) функции, могут быть определены как статические. Если переменная, объявленная на локальном уровне не является статической, то распределение памяти под такую переменную производится автоматически.

**Пример:**

```
int Counter()
{
    static int count;
    count++;
    if(count%100==0)
        Print("Функция Counter была вызвана уже ",count," раз");
    return
        count;
}
void OnStart()
{
//---
    int c=345;
    for(int i=0;i<1000;i++)
        int c=Counter();
    Print("c = ",c);
}
```

**Смотри также**

[Типы данных](#), [Инкапсуляция и расширяемость типов](#), [Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#), [Статические члены класса](#)

## Глобальные переменные

Глобальные переменные создаются путем размещения их объявлений вне описания какой-либо функции. 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 OnStart()
{
    ...
}
```

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 либо константным выражением.

Инициализация глобальных переменных производится однократно после загрузки программы в память клиентского терминала и перед первой обработкой события [Init](#). Для глобальных переменных, представляющих собой объекты классов, при инициализации вызываются соответствующие конструкторы. В скриптах инициализация глобальных переменных производится перед обработкой события [Start](#).

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

**Смотри также**

[Типы данных](#), [Инкапсуляция и расширяемость типов](#), [Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

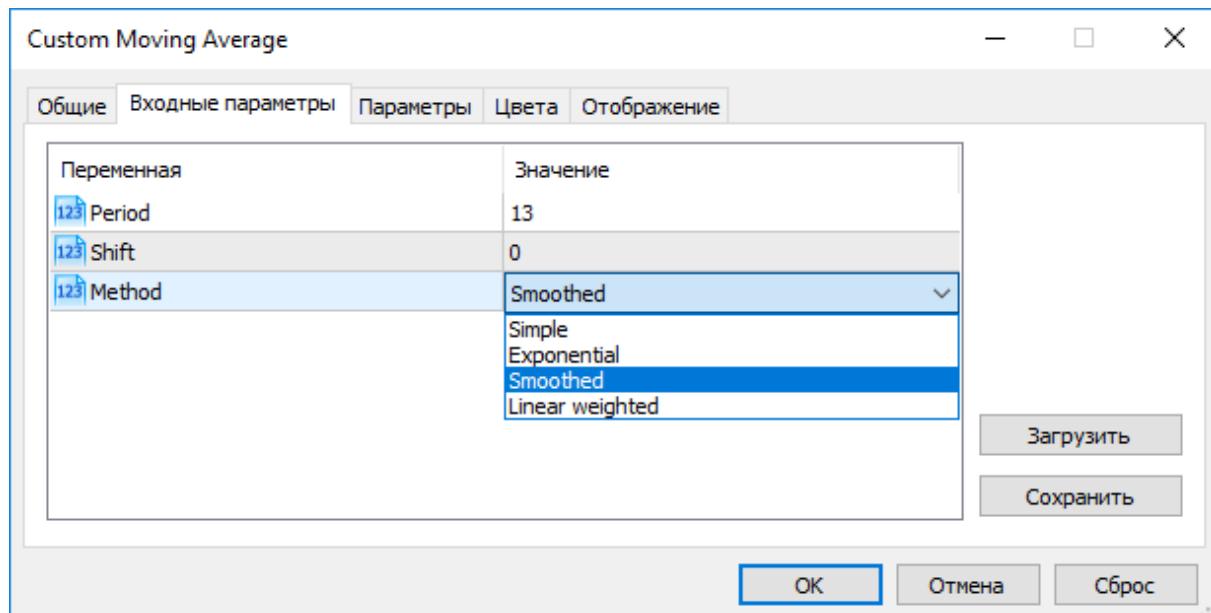
## Input переменные

Класс памяти `input` определяет внешнюю переменную. Модификатор `input` указывается перед типом данных. Изменять значение переменной с модификатором `input` внутри mq5-программы нельзя, такие переменные доступны только для чтения. Изменять значения `input`-переменных может только пользователь из окна свойств программы. Внешние переменные всегда переинициализируются непосредственно перед вызовом `OnInit()`.

**Пример:**

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

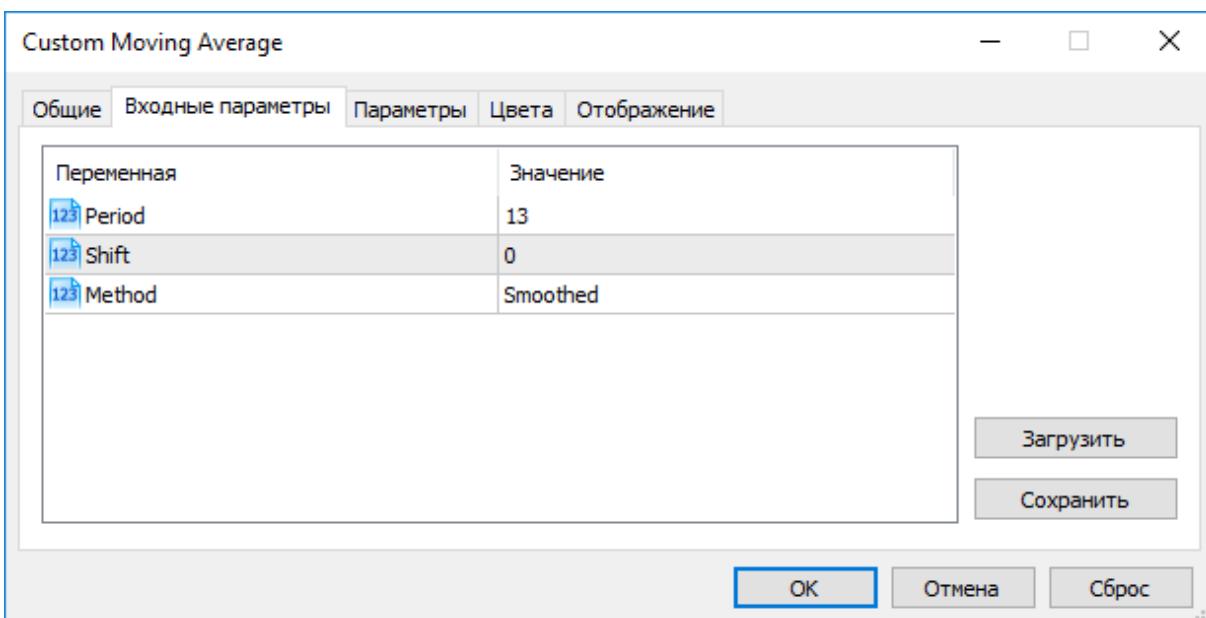
Input переменные определяют входные параметры программы, они доступны из окна свойств программы.



Существует возможность задать иной способ отображения имен входных параметров на закладке "Inputs". Для этого используется строчный комментарий, который должен располагаться после описания входного параметра в той же строке. Таким образом, входным параметрам можно сопоставить более понятные для пользователя имена.

**Пример:**

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



**Примечание:** Массивы и переменные [сложных типов](#) не могут выступать в качестве input-переменных.

**Примечание:** Длина строчного комментария для Input переменных не может превышать 63 символа.

## Передача параметров при вызове пользовательских индикаторов из mq5-программ

Пользовательские индикаторы вызываются при помощи функции [iCustom\(\)](#). При этом после имени пользовательского индикатора должны идти параметры в точном соответствии с объявлением input-переменных данного пользовательского индикатора. Если параметров указывается меньше, чем объявлено input-переменных в вызываемом пользовательском индикаторе, то недостающие параметры заполняются значениями, указанными при объявлении переменных.

Если в пользовательском индикаторе используется функция [OnCalculate](#) первого вида (то есть, индикатор считается на одном массиве данных), то в качестве последнего параметра при вызове такого пользовательского индикатора должно выступать одно из значений [ENUM\\_APPLIED\\_PRICE](#) либо хэндл другого индикатора. При этом все параметры, соответствующие input-переменным, должны быть явно указаны.

### Перечисления в качестве input-параметра

В качестве input-переменных (входных параметров для mq5-программ) можно использовать не только предусмотренные языком MQL5 встроенные перечисления, но и перечисления, заданные пользователем. Например, мы можем создать перечисление `dayOfWeek`, описывающее дни недели, и использовать input-переменную для указания конкретного дня недели не в виде цифры, а в более привычном для пользователя виде.

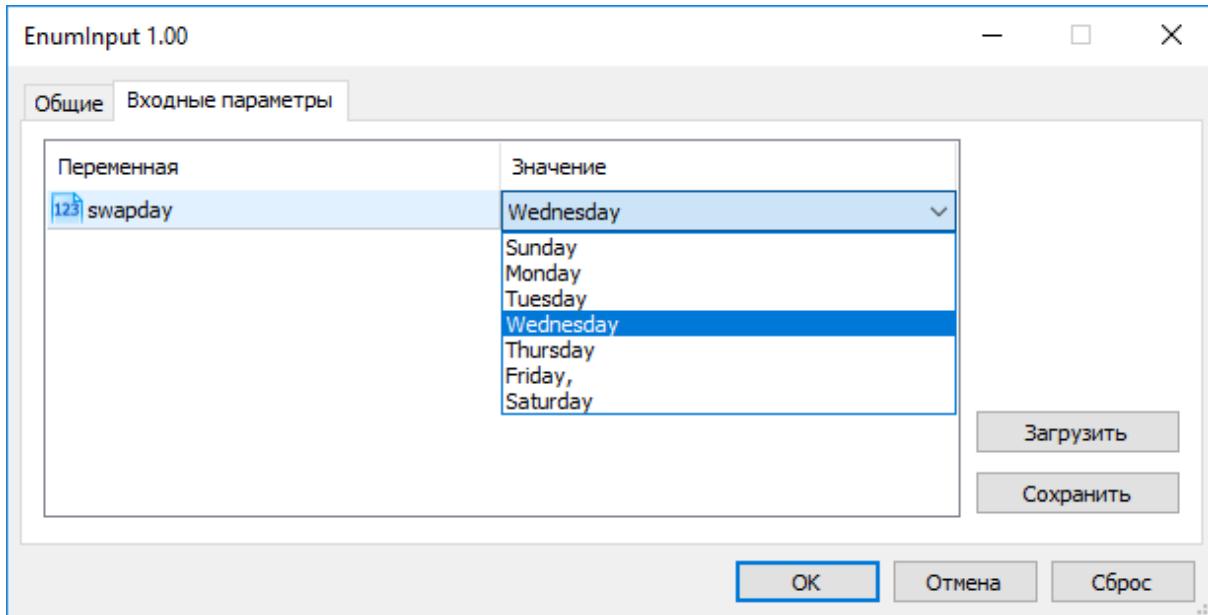
**Пример:**

```
#property script_show_inputs
```

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

//--- input parameters
input dayOfWeek swapday=W;
```

Для того чтобы при запуске скрипта пользователь мог выбрать нужное значение из окна свойств, мы используем команду препроцессора `#property script_show_inputs`. Запускаем скрипт на исполнение и можем выбрать из списка одно из значений перечислений `dayOfWeek`. Запускаем скрипт `EnumInInput` и переходим на закладку "Параметры". По умолчанию, значение параметра `swapday` (день начисления тройного свопа) является среда (`W=3`), но мы можем задать любое другое значение и использовать это значение для изменения работы программы.



Количество возможных значений перечисления ограничено. Поэтому для выбора входного значения используется выпадающий список. В качестве значений, показываемых в списке, используются мнемонические имена членов перечисления. Если же мнемоническому имени сопоставлен комментарий, как это показано в нашем примере, то вместо мнемонического имени используется содержимое комментария.

Каждое значение из перечисления `dayOfWeek` имеет свое значение от 0 до 6, но в списке параметров будут показаны комментарии, указанные для каждого значения. Это дает дополнительную гибкость для написания программ с понятными описаниями входных параметров.

## Переменные с модификатором `sinput`

Переменные с модификатором `input` позволяют не только задавать значения внешних параметров при запуске программ, но также играют большую роль при оптимизации торговых стратегий в тестере. Каждая объявленная в эксперте `input`-переменная, за исключением типа `string`, может участвовать в оптимизации.

В некоторых случаях бывает необходимо исключить некоторые внешние параметры программы из формирования области всех возможных проходов в тестере. Специально для таких случаев существует модификатор памяти `sinput`. `sinput` - это сокращенное написание объявления статической внешней переменной: `sinput = static input`. То есть такое объявление в коде советника

```
sinput      int layers=6; // Количество слоев
```

будет эквивалентно полному объявлению

```
static input int layers=6; // Количество слоев
```

Переменная, объявленная с модификатором `sinput`, является входным параметром MQL5-программы, значение этого параметра можно изменять при её запуске. Но при этом данная переменная не участвует в процессе оптимизации входных параметров, то есть не производится перебор её значений при поиске наилучшего набора параметров по заданному критерию.

Тестер стратегий						
Переменная	Значение	Старт	Шаг	Стоп	Шаги	
<input type="checkbox"/> Количество слоев	6					
<input checked="" type="checkbox"/> Нейронов в слое	30	30	1	300	271	
<input checked="" type="checkbox"/> Количество баров для анализа	13	13	1	130	118	
<input checked="" type="checkbox"/> Горизонт прогноза	2	2	1	20	19	
<input type="checkbox"/> Тип сети	0	0	1	10		
					607582	

[Настройки](#) [Параметры](#) [Оптимизация](#) | [Агенты](#) | [Журнал](#) |

На рисунке показано, что эксперт имеет 5 внешних параметров, из них параметр "Количество слоев" объявлен как `sinput` и равен 6. Этот параметр не может изменяться в процедуре оптимизации торговой стратегии, для него возможно установить нужное значение, которое и будет использоваться. Поля Старт, Шаг и Стоп для такой переменной не доступны для установки значений.

Таким образом, задав для переменной модификатор `sinput`, мы запрещаем пользователю оптимизировать данный параметр. Это значит, что в тестере стратегий пользователю терминала становится недоступным задавать для неё начальное и конечное значения для автоматического перебора в указанном диапазоне в процессе оптимизации.

Но при этом есть одно исключение из данного правила - `sinput`-переменные можно варьировать в задачах оптимизации с помощью функции `ParameterSetRange()`. Данная функция создана специально для программного управления пространством доступных значений для любых `input`-переменных, в том числе и объявленных как `static input` (`sinput`). Другая функция `ParameterGetRange()` позволяет при запуске оптимизации (в обработчике `OnTesterInit()`) получить значения `input`-переменных и в случае необходимости переопределить шаг изменения и диапазон, в пределах которого будет перебираться значение оптимизируемого параметра.

Таким образом, сочетание модификатора `sinput` и двух функций по работе с `iinput`-параметрами позволяет создавать гибкие правила для задания интервалов оптимизации одних `iinput`-переменных в зависимости от значения других `iinput`-переменных.

#### Смотри также

[iCustom](#), [Перечисления](#), [Свойства программ](#)

## Extern переменные

Ключевое слово `extern` используется, чтобы объявить идентификаторы переменных как идентификаторы [статического класса памяти](#) с глобальным [временем жизни](#). Такие переменные существуют с момента начала выполнения программы и для них память выделяется и инициализируется сразу после начала выполнения программы.

Можно создавать программы, которые состоят из нескольких исходных файлов, для этого используется директива препроцессора `#include`. Переменные, объявленные как `extern` с одним и тем же типом и идентификатором, могут существовать в разных исходных файлах одного проекта.

При компиляции всего проекта все `extern`-переменные с одним и тем же типом и идентификатором ассоциируются с одним участком памяти пула глобальных переменных. `Extern`-переменные полезны для раздельной компиляции исходных файлов. `Extern`-переменные можно инициализировать, но только однократно - недопустимо существование нескольких инициализированных `extern`-переменных одного и того же типа и с одним и тем же идентификатором.

### Смотри также

[Типы данных](#), [Инкапсуляция и расширяемость типов](#), [Инициализация переменных](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Инициализация переменных

Любая переменная при определении может быть инициализирована. Если не произведена явная инициализация переменной, то значение, хранящееся в данной переменной, может быть каким угодно. Неявная инициализация не производится.

Глобальные и статические переменные могут быть проинициализированы только константой соответствующего типа или константным выражением. 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;
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} };
//--- из тетриса
int      right[4]={WIDTH_IN_PIXELS+VERT_BORDER,WIDTH_IN_PIXELS+VERT_BORDER,
                  WIDTH_IN_PIXELS+VERT_BORDER,WIDTH_IN_PIXELS+VERT_BORDER};
//--- инициализация всех полей структуры нулевым значением
MqlTradeRequest request={0};
```

Список значений элементов массива должен быть заключен в фигурные скобки. Пропущенные инициализирующие последовательности считаются равными 0. В инициализирующей последовательности должно быть хотя бы одно значение: этим значением инициализируется первый элемент соответствующей структуры или массива, отсутствующие элементы считаются равными нулю.

Если размер инициализируемого массива не указан, то он определяется компилятором, исходя из размера инициализирующей последовательности. Многомерные массивы не могут инициализироваться одномерной последовательностью (последовательностью без дополнительных фигурных скобок), за исключением одного случая - когда указывается всего один инициализирующий элемент (как правило, ноль).

Массивы (в том числе и объявленные на локальном уровне) могут инициализироваться только константами.

**Примеры:**

```
struct str3
{
    int           low_part;
    int           high_part;
};

struct str10
{
    str3          s3;
    double        d1[10];
    int           i3;
```

```
};

void OnStart()
{
    str10 s10_1={{1,0},{1.0,2.1,3.2,4.4,5.3,6.1,7.8,8.7,9.2,10.0},100};
    str10 s10_2={{1,0},{0},100};
    str10 s10_3={{1,0},{1.0}};

//---
    Print("1. s10_1.d1[5] = ",s10_1.d1[5]);
    Print("2. s10_2.d1[5] = ",s10_2.d1[5]);
    Print("3. s10_3.d1[5] = ",s10_3.d1[5]);
    Print("4. s10_3.d1[0] = ",s10_3.d1[0]);
}
```

Для переменных типа структур допускается частичная инициализация, это же относится и к статическим массивам (с явно заданным размером). Можно проинициализировать один или несколько первых элементов структуры или массива, оставшиеся элементы в таком случае будут проинициализированы нулями.

#### Смотри также

[Типы данных](#), [Инкапсуляция и расширяемость типов](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## Область видимости и время жизни переменных

Существует два основных вида области видимости: [локальная](#) область видимости и [глобальная](#) область видимости.

Переменная, объявленная вне всех функций, помещается в глобальную область видимости. Доступ к таким переменным может осуществляться из любого места программы. Такие переменные располагаются в глобальном пуле памяти, поэтому время их жизни совпадает со временем жизни программы.

Переменная, объявленная внутри блока (часть кода, заключенная в фигурные скобки), принадлежит локальной области видимости. Такая переменная не видна (поэтому и недоступна) за пределами блока, в котором она объявлена. Самый распространенный случай локального объявления - переменная, объявленная внутри функции. Переменная, объявленная локально, располагается на стеке, и время жизни такой переменной совпадает со временем жизни функции.

Так как областью видимости локальной переменной является блок, в котором она объявлена, то существует возможность объявлять переменные с именем, совпадающим с именами переменных, объявленных в других блоках; а также объявленных на более верхних уровнях, вплоть до глобального.

**Пример:**

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

```

```

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

```

Обратите внимание на переменную `i`, объявленную в строке

```

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

```

Ее область видимости - только цикл `for`, за пределами этого цикла действует другая переменная с тем же именем, объявленная в начале функции. Кроме того, в теле цикла объявлена переменная `k`, областью видимости которой является тело цикла.

Локальные переменные можно объявлять со спецификатором доступа [static](#). В этом случае компилятор располагает такую переменную в глобальном пуле памяти. Поэтому, время жизни статической переменной совпадает со временем жизни программы. При этом область видимости такой переменной ограничивается пределами блока, в котором она объявлена.

#### Смотри также

[Типы данных](#), [Инкапсуляция и расширяемость типов](#), [Инициализация переменных](#), [Создание и уничтожение объектов](#)

## Создание и уничтожение объектов

После загрузки на исполнение mq5-программы каждой переменной выделяется память в соответствие с типом переменной. Переменные делятся на два типа по уровню доступа - глобальные переменные и локальные переменные, и по классам памяти: входные параметры mq5-программы, статические и автоматические. Каждая переменная при необходимости инициализируется соответствующим значением. После использования переменная деинициализируется и память, использованная ею, возвращается исполняемой системе MQL5.

## Инициализация и деинициализация глобальных переменных

Инициализация глобальных переменных производится автоматически сразу после загрузки mq5-программы и до вызова любой функции. При инициализации производится присвоение начальных значений переменным простых типов и вызывается конструктор для объектов, если он есть. Входные переменные всегда объявляются на глобальном уровне, инициализируются значениями, задаваемыми пользователями в диалоге при запуске mq5-программы.

Несмотря на то, что статические переменные обычно объявляются на локальном уровне, память под эти переменные распределяется заранее, и инициализация производится сразу после загрузки программы, точно так же как и для глобальных переменных.

Порядок инициализации соответствует порядку объявления переменной в программе, а деинициализация производится в обратном порядке перед выгрузкой mq5-программы. Это правило только для тех переменных, которые не были созданы оператором new. Такие переменные создаются и инициализируются автоматически сразу после загрузки, а деинициализируются непосредственно перед выгрузкой программы.

## Инициализация и деинициализация локальных переменных

Если переменная, объявленная на локальном уровне, не является статической, то распределение памяти под такую переменную производится автоматически. Локальные переменные, также как и глобальные, инициализируются автоматически в тот момент, когда выполнение программы встречает объявление локальной переменной. Таким образом, порядок инициализации соответствует порядку объявления.

Локальные переменные деинициализируются в конце блока программы, в котором они объявлены, и в порядке, обратном их объявлению. Блок программы - это составной оператор, который может являться частью оператора выбора switch, цикла(for, while, do-while), телом функции или частью оператора if-else.

Инициализация локальных переменных происходит только в тот момент, когда выполнение программы доходит до объявления переменной. Если в процессе выполнения программы блок, в котором объявлена переменная, не был выполнен, то такая переменная не инициализируется.

## Инициализация и деинициализация динамически размещаемых объектов

Особый случай представляют из себя указатели объектов, так как объявление указателя не влечет за собой инициализации соответствующего объекта. Динамически размещаемые объекты инициализируются только в момент создания экземпляра класса оператором new. Инициализация

объекта предполагает вызов конструктора соответствующего класса. Если соответствующего конструктора в классе нет, то его члены, имеющие [простой тип](#), не будут автоматически проинициализированы; члены типов [строка](#), [динамический массив](#) и [сложный объект](#) будут автоматически проинициализированы.

Указатели могут быть объявлены на локальном или глобальном уровне и при этом могут быть проинициализированы пустым значением [NULL](#) или значением указателя такого же или [порожденного](#) типа. Если для указателя, объявленного на локальном уровне, был вызван оператор *new*, то и оператор *delete* для этого указателя должен быть выполнен до выхода из этого уровня. В противном случае указатель будет потерян, и объект не сможет быть удален явно.

Все объекты, созданные выражением [указатель\\_объекта=new Имя\\_Класса](#), обязательно должны быть впоследствии уничтожены оператором *delete*([указатель\\_объекта](#)). Если по каким-то причинам такая переменная по окончании работы программы не была уничтожена [оператором delete](#), то об этом будет выведено сообщение в журнал "Эксперты". Можно объявить несколько переменных и всем им присвоить указатель одного объекта.

Если динамически создаваемый объект имеет конструктор, то этот конструктор будет вызван в момент выполнения оператора *new*. Если объект имеет деструктор, то деструктор будет вызван в момент выполнения оператора *delete*.

Таким образом, динамически размещаемые объекты создаются только в момент создания оператором *new* и гарантированно уничтожаются либо оператором *delete*, либо автоматически исполняющей системой MQL5 в момент выгрузки программы. Порядок объявления указателей динамически создаваемых объектов не влияет на порядок их инициализации. Порядок инициализации и деинициализации полностью контролируется программистом.

## Особенности работы с динамической памятью

При работе с динамическими массивами освобожденная память сразу же возвращается в систему.

При создании динамического объекта класса через [new](#), память сначала ищется в пуле памяти классов, с которым работает менеджер памяти, и если в пуле недостаточно памяти, то память запрашивается в системе. При удалении динамического объекта через [delete](#), память, занимаемая объектом, возвращается в пул памяти классов.

Менеджер памяти возвращает память в систему сразу после выхода из функций-обработчиков событий: [OnInit\(\)](#), [OnDeinit\(\)](#), [OnStart\(\)](#), [OnTick\(\)](#), [OnCalculate\(\)](#), [OnTimer\(\)](#), [OnTrade\(\)](#), [OnTester\(\)](#), [OnTesterInit\(\)](#), [OnTesterPass\(\)](#), [OnTesterDeinit\(\)](#), [OnChartEvent\(\)](#), [OnBookEvent\(\)](#).

## Краткая характеристика переменных

Основные сведения о порядке создания, уничтожении, вызове конструкторов и деструкторов приведены в краткой таблице.

	Глобальная автоматическая переменная	Локальная автоматическая переменная	Динамически создаваемый объект
Инициализация	сразу после загрузки mql5-программы	при достижении в ходе выполнения	при выполнении оператора <i>new</i>

		строки кода, где она объявлена	
<b>Порядок инициализации</b>	в порядке объявления	в порядке объявления	не зависит от порядка объявления
<b>Деинициализация</b>	перед выгрузкой mql5-программы	при выходе выполнения из блока объявления	при выполнении оператора <code>delete</code> или перед выгрузкой mql5-программы
<b>Порядок деинициализации</b>	в порядке, обратном инициализации	в порядке, обратном инициализации	не зависит от порядка инициализации
<b>Вызов конструктора</b>	при загрузке mql5-программы	при инициализации	при выполнении оператора <code>new</code>
<b>Вызов деструктора</b>	при выгрузке mql5-программы	при выходе из блока, в котором переменная была инициализирована	при выполнении оператора <code>delete</code>
<b>Сообщения об ошибках</b>	сообщение в журнал "Эксперты" о попытке удаления автоматически созданного объекта	сообщение в журнал "Эксперты" о попытке удаления автоматически созданного объекта	сообщение в журнал "Эксперты" о неудаленных динамически созданных объектах при выгрузке mql5-программы

Смотри также

[Типы данных](#), [Инкапсуляция и расширяемость типов](#), [Инициализация переменных](#), [Область видимости и время жизни переменных](#)

## Препроцессор

Препроцессор - это специальная подсистема компилятора MQL5, 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 mqf5-программ. Улучшению читаемости кода способствует и возможность присвоения мнемонических имен отдельным константам.

Препроцессор позволяет также определять специфические параметры mqf5-программ:

- [Объявлять константы](#)
- [Устанавливать свойства программы](#)
- [Включать в текст программы файлы](#)
- [Импортировать функции](#)
- [Использовать условную компиляцию](#)

Директивы препроцессора используются компилятором для предварительной обработки исходного кода перед его компиляцией. Директива всегда начинается со знака # (решетка), поэтому компилятор запрещает использовать данный символ в именах переменных, функций и т.д.

Каждая директива описывается отдельной записью и действует до переноса строки. Нельзя в одной записи использовать несколько директив. Если запись директивы получается слишком большой, то её можно разбить на несколько строк с помощью обратного слеша \, в таком случае следующая строка будет считаться продолжением записи директивы.

```
//+-----+
//| псевдооператор foreach
//+-----+
#define ForEach(index, array) for (int index = 0, \
    max_##index=ArraySize((array)); \
    index<max_##index; index++)
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    string array[]={ "12", "23", "34", "45" };
//--- обход массива с помощью ForEach
    ForEach(i,array)
    {
        PrintFormat ("%d: array[%d]=%s",i,i,array[i]);
    }
}
//+-----+
/* Результат вывода
0: array[0]=12
1: array[1]=23
```

```
2: array[2]=34  
3: array[3]=45  
*/
```

Для компилятора все эти три строчки директивы `#define` будут выглядеть как одна длинная строка. В этом примере также используется двойной знак решетки `##`, который называется оператором слияния и используется в макросах `#define` для объединения двух токенов макроса в один. Оператор слияния токенов не может быть первым или последним в определении макроса.

## Макроподстановка (`#define`, `#undef`)

Директивы препроцессора используются компилятором для предварительной обработки исходного кода перед его компиляцией. Директива всегда начинается со знака `#` (решетка), поэтому компилятор запрещает использовать данный символ в именах переменных, функций и т.д.

Каждая директива описывается отдельной записью и действует до переноса строки. Нельзя в одной записи использовать несколько директив. Если запись директив получается слишком большой, то её можно разбить на несколько строк с помощью обратного слеша `\`, в таком случае следующая строка будет считаться продолжением записи директивы.

Директива `#define` может быть использована для присвоения мнемонических имен выражениям. Существует две формы:

<code>#define identifier expression</code>	<code>// беспараметрическая форма</code>
<code>#define identifier(par1,... par8) expression</code>	<code>// параметрическая форма</code>

Директива `#define` подставляет `expression` вместо всех последующих найденных вхождений `identifier` в исходном тексте. `identifier` заменяется только в том случае, если он представляет собой отдельный токен. `identifier` не заменяется, если он является частью комментария, частью строки, или частью другого более длинного идентификатора.

Идентификатор константы подчиняется тем же правилам, что и для имен переменных. Значение может быть любого типа:

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

`expression` может состоять из нескольких токенов, таких как ключевые слова, константы, константные и неконстантные выражения. `expression` заканчивается с концом строки и не может быть перенесено на следующую строку.

**Пример:**

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

```
(2 + 3)*2 = 10
*/
```

## Параметрическая форма #define

При параметрической форме все последующие найденные вхождения `identifier` будут заменены на `expression` с учетом фактических параметров. Например,

```
// пример с двумя параметрами а и б
#define A 2+3
#define B 5-1
#define MUL(a, b) ((a)*(b))

double c=MUL(A,B);
Print("c=",c);
/*
выражение double c=MUL(A,B);
равносильно double c=((2+3)*(5-1));
*/
// Результат
// c=20
```

Обязательно заключайте параметры в круглые скобки при использовании параметров в `expression`, так как это позволит избежать неочевидных ошибок, которые трудно найти. Если переписать пример без использования скобок, то результат окажется совсем другой:

```
// пример с двумя параметрами а и б
#define A 2+3
#define B 5-1
#define MUL(a, b) a*b

double c=MUL(A,B);
Print("c=",c);
/*
выражение double c=MUL(A,B);
равносильно double c=2+3*5-1;
*/
// Результат
// c=16
```

При использовании параметрической формы допускается не более 8 параметров.

```
// правильная параметрическая форма
#define LOG(text) Print(__FILE__,"(",__LINE__,") :",text) // один параметр - 'text'

// неправильная параметрическая форма
#define WRONG_DEF(p1, p2, p3, p4, p5, p6, p7, p8, p9) p1+p2+p3+p4 // более 8 параметров
```

## Директива #undef

Директива `#undef` предназначена для отмены макроса, объявленного ранее.

**Пример:**

```
#define MACRO

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

#undef MACRO

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

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

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

**Смотри также**

[Идентификаторы](#), [Символьные константы](#)

## Свойства программ (#property)

У каждой mq5-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. Свойства, описанные во включаемых файлах, полностью игнорируются. Свойства необходимо задавать в главном mq5-файле.

```
#property идентификатор значение
```

Компилятор запишет в настройках выполняемого модуля объявленные значения.

Константа	Тип	Описание
icon	<a href="#">string</a>	Путь к файлу с картинкой, которая будет показываться для программы EX5. Правила указания пути такие же, как и для <a href="#">ресурсов</a> . Свойство должно указываться в главном модуле с исходным кодом MQL5. Файл иконки должен быть в формате <a href="#">ICO</a> .
link	<a href="#">string</a>	Ссылка на сайт компании-производителя
copyright	<a href="#">string</a>	Название компании-производителя
version	<a href="#">string</a>	Версия программы, не более 31 символа
description	<a href="#">string</a>	Краткое текстовое описание mq5-программы. Может присутствовать несколько description, каждый из которых описывает одну строку текста. Общая длина всех description не может превышать 511 символов с учетом переводов строк
stacksize	<a href="#">int</a>	Указывает размер <a href="#">стека</a> для MQL5 программы, стек достаточного объема требуется в случае выполнения рекурсивных вызовов функций. При запуске скрипта или эксперта на графике выделяется стек не менее 8Мб, для индикаторов свойство не работает - стек

		всегда фиксированного объема в 1Мб. При запуске в тестере программе всегда выделяется стек в размере 16 Мб.
library		Библиотека; не назначается никакой стартовой функции, функции с <a href="#">модификатором export</a> можно <a href="#">импортировать</a> в других mql5-программах
indicator_applied_price	<a href="#">int</a>	Задает значение по умолчанию для поля <a href="#">"Apply to"</a> . Можно задавать одно из значений перечисления <a href="#">ENUM_APPLIED_PRICE</a> . Если свойство не задано, то по умолчанию применяется значение PRICE_CLOSE
indicator_chart_window		Выводить индикатор в окно графика
indicator_separate_window		Выводить индикатор в отдельное окно
indicator_height	<a href="#">int</a>	Фиксированная высота подокна индикатора в пикселях (свойство <a href="#">INDICATOR_HEIGHT</a> )
indicator_buffers	<a href="#">int</a>	Количество буферов для расчета индикатора
indicator_plots	<a href="#">int</a>	Количество <a href="#">графических серий</a> в индикаторе
indicator_minimum	<a href="#">double</a>	Нижнее ограничение шкалы отдельного окна индикатора
indicator_maximum	<a href="#">double</a>	Верхнее ограничение шкалы отдельного окна индикатора
indicator_labelN	<a href="#">string</a>	Задает метку для N-ой <a href="#">графической серии</a> , отображаемую в окне DataWindow. Для графических серий, требующих нескольких индикаторных буферов (DRAW_CANDLES, DRAW_FILLING и другие), имена меток задаются через разделитель ':'.

indicator_colorN	<a href="#">color</a>	Цвет для вывода линии N, где N - номер <a href="#">графической серии</a> ; нумерация с 1
indicator_widthN	<a href="#">int</a>	Толщина линии в <a href="#">графической серии</a> , где N - номер графической серии; нумерация с 1
indicator_styleN	<a href="#">int</a>	Стиль линии в <a href="#">графической серии</a> , указываемый с помощью значения из <a href="#">ENUM_LINE_STYLE</a> . N - номер графической серии, нумерация с 1
indicator_typeN	<a href="#">int</a>	Вид графического построения, указываемый с помощью значения из <a href="#">ENUM_DRAW_TYPE</a> . N - номер графической серии, нумерация с 1
indicator_levelN	<a href="#">double</a>	Горизонтальный уровень N в отдельном окне индикатора
indicator_levelcolor	<a href="#">color</a>	Цвет горизонтальных уровней индикатора
indicator_levelwidth	<a href="#">int</a>	Толщина горизонтальных уровней индикатора
indicator_levelstyle	<a href="#">int</a>	Стиль горизонтальных уровней индикатора
script_show_confirm		Выводить окно подтверждения перед запуском скрипта
script_show_inputs		Выводить окно со свойствами перед запуском скрипта и запретить вывод окна подтверждения
tester_indicator	<a href="#">string</a>	Имя пользовательского индикатора в формате "имя_индикатора.ex5". Необходимые для тестирования индикаторы определяются автоматически из вызова функций <a href="#">iCustom()</a> , если соответствующий параметр задан константной строкой. Для остальных случаев (использование функции <a href="#">IndicatorCreate()</a> ) или

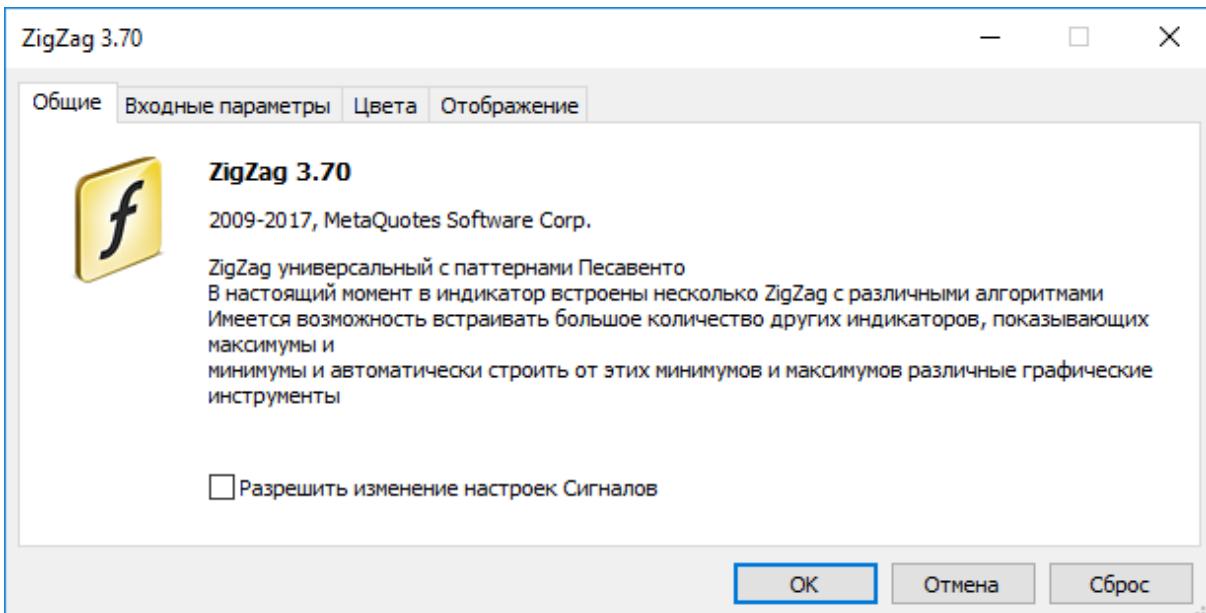
		использование неконстантной строки в параметре, задающем имя индикатора) необходимо данное свойство
tester_file	<a href="#">string</a>	Имя файла для тестера с указанием расширения, заключенное в двойные кавычки (как константная строка). Указанный файл будет передан тестеру в работу. Входные файлы для тестирования, если необходимы, должны указываться всегда
tester_library	<a href="#">string</a>	Имя библиотеки с расширением, заключенное в двойные кавычки. Библиотека может быть как с расширением dll, так и с расширением ex5. Необходимые для тестирования библиотеки определяются автоматически. Однако если какая-либо библиотека используется <a href="#">пользовательским</a> индикатором, то необходимо использовать данное свойство
tester_set	<a href="#">string</a>	Имя set-файла со значениями и шагом входных параметров. Указанный файл будет передан тестеру в работу перед началом тестирования или оптимизации. Имя файла необходимо указывать с расширением и в двойных кавычках как константную строку.  Если в названии set-файла указать имя эксперта и номер версии как "<expert_name>_<number>.set", то он автоматически добавится в меню загрузки версий параметров под номером версии <number>. Например, имя "MACD Sample_4.set" означает, что это set-файл для эксперта "MACD Sample.mq5" с номером версии равным 4.

		Для изучения формата рекомендуем вручную сохранить настройки тестирования/оптимизации в тестере стратегий и затем открыть созданный таким образом set-файл.
tester_no_cache	<a href="#">string</a>	<p>Тестер стратегий при выполнении <a href="#">оптимизации</a> сохраняет все результаты выполненных проходов в <a href="#">кеш оптимизации</a>, в котором для каждого набора <a href="#">входных параметров</a> сохраняется результат тестирования. Это позволяет при повторной оптимизации на тех же параметрах брать готовые результаты без повторного вычисления и затрат времени.</p> <p>Но для некоторых задач - например, при математических вычислениях - может потребоваться проводить расчеты независимо от наличия готовых результатов в кеше оптимизации. В этом случае в файл необходимо включить свойство <i>tester_no_cache</i>. При этом сами результаты тестирования все равно будут сохраняться в кеше, чтобы можно было в тестере стратегий посмотреть все данные по выполненным проходам.</p>
tester_evertick_calculate	<a href="#">string</a>	<p>В тестере стратегий индикаторы рассчитываются только при обращении к ним за данными - то есть только в тот момент, когда запрашиваются значения индикаторных буферов. Это даёт существенное ускорение при тестировании и оптимизации, если не требуется получать значения индикатора на каждом тике.</p> <p>Указание свойства <i>tester_evertick_calculate</i> позволяет при тестировании</p>

		<p>принудительно включить режим расчета индикатора на <a href="#">каждом тике</a>.</p> <p>Индикаторы в тестере стратегий также принудительно считаются на каждом тике в следующих случаях:</p> <ul style="list-style-type: none"> <li>• при тестировании в <a href="#">визуальном режиме</a>;</li> <li>• при наличии в индикаторе функций <a href="#">EventChartCustom</a>, <a href="#">OnChartEvent</a>, <a href="#">OnTimer</a>;</li> <li>• если индикатор создан компилятором с <a href="#">номером билда</a> ниже 1916.</li> </ul> <p>Данное свойство касается только работы в тестере стратегий, в терминале индикаторы всегда считаются на каждом поступившем тике.</p>
--	--	---

### Пример задания описания и номера версии

```
#property version      "3.70"          // текущая версия эксперта
#property description "ZigZag универсальный с паттернами Песавенто"
#property description "В настоящий момент в индикатор встроены несколько ZigZag с различными алгоритмами"
#property description "Имеется возможность встраивать большое количество других индикаторов, показывающих максимумы и минимумы и автоматически строить от этих минимумов и максимумов различные графические инструменты"
#property description "минимумы и автоматически строить от этих минимумов и максимумов различные графические инструменты"
```



Пример указания отдельной метки для каждого индикаторного буфера ("C open;C high;C low;C close")

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



## Включение файлов (#include)

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\МQL5\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.

Смотри также

[Стандартная библиотека](#), [Импорт функций](#)

## Импорт функций (#import)

Импорт функций осуществляется из откомпилированных модулей MQL5 (файлы \*.ex5) 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 могут иметь любые имена. Можно одновременно импортировать из разных модулей функции с одинаковыми именами. Импортируемые функции могут иметь имена, совпадающие с именами встроенных функций. Операция [разрешения контекста](#) определяет, какая из функций должна вызываться.

Порядок поиска файла, указанного после ключевого слова `#import`, описан в разделе [Вызов импортируемых 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 EX5, так и из DLL-модулей), могут иметь значения по умолчанию.

В импортируемых функциях в качестве параметров нельзя использовать:

- [указатели \(\\*\)](#);
- ссылки на объекты, содержащие [динамические массивы](#) и/или указатели.

В импортируемые из DLL функции нельзя передавать в качестве параметра классы, массив строк или сложные объекты, содержащие строки и/или динамические массивы любых типов.

**Примеры:**

```
#import "stdlib.ex5"
string ErrorDescription(int error_code);
int     RGB(int red_value,int green_value,int blue_value);
bool    CompareDoubles(double number1,double number2);
string  DoubleToStrMorePrecision(double number,int precision);
string  IntegerToHexString(int integer_number);
#import "ExpertSample.dll"
int     GetIntValue(int);
double  GetDoubleValue(double);
string  GetStringValue(string);
double  GetArrayItemValue(double &arr[],int,int);
bool    SetArrayItemValue(double &arr[],int,int,double);
double  GetRatesItemValue(double &rates[][6],int,int,int);
```

```
#import
```

Для импорта функций во время выполнения MQL5-программы используется раннее связывание. Это значит, что библиотека загружается в процессе загрузки использующей ее EXE-программы.

Не рекомендуется использовать полностью квалифицированное имя загружаемого модуля вида *Drive:\Directory\FileName.Ext*. Библиотеки MQL5 загружаются из папки *terminal\_dir\MQL5\Libraries*.

Если импортируемая функция имеет разные варианты вызова для 32-х и 64-х битной версий Windows, то необходимо импортировать оба и явно вызывать нужный вариант функции с помощью переменной [\\_IsX64](#).

**Пример:**

```
#import "user32.dll"
//--- для 32-х битной системы
int MessageBoxW(uint hWnd, string lpText, string lpCaption, uint uType);
//--- для 64-х битной системы
int MessageBoxW(ulong hWnd, string lpText, string lpCaption, uint uType);
#import
//+-----+
//| MessageBox_32_64_bit использует нужный вариант MessageBoxW() |
//+-----+
int MessageBox_32_64_bit()
{
    int res=-1;
    //--- если у нас 64-битная Windows
    if(_IsX64)
    {
        ulong hwnd=0;
        res=MessageBoxW(hwnd, "Пример вызова 64-битной версии MessageBoxW", "MessageBoxW");
    }
    else // у нас 32-битная Windows
    {
        uint hwnd=0;
        res=MessageBoxW(hwnd, "Пример вызова 32-битной версии MessageBoxW", "MessageBoxW");
    }
    return (res);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//---
    int ans=MessageBox_32_64_bit();
    PrintFormat("MessageBox_32_64_bit returned %d",ans);
}
```

Смотри также

[Включение файлов](#)

## Условная компиляция (#ifdef, #ifndef, #else, #endif)

Директивы препроцессора используются компилятором для предварительной обработки исходного кода перед его компиляцией. Директива всегда начинается со знака `#` (решетка), поэтому компилятор запрещает использовать данный символ в именах переменных, функций и т.д.

Каждая директива описывается отдельной записью и действует до переноса строки. Нельзя в одной записи использовать несколько директив. Если запись директив получается слишком большой, то её можно разбить на несколько строк с помощью обратного слеша `\`, в таком случае следующая строка будет считаться продолжением записи директивы.

Директивы условной компиляции препроцессора позволяют компилировать или пропускать часть программы в зависимости от выполнения некоторого условия.

Условие может принимать одну из описываемых ниже форм.

```
#ifdef identifier
    // код, находящийся здесь, компилируется, если identifier уже был определен для пре
#endif
#ifndef identifier
    // код, находящийся здесь, компилируется, если identifier в данный момент не опреде
#endif
```

За любой из команд условной компиляции может следовать произвольное число строк, содержащих, возможно, команду вида `#else` и заканчивающихся `#endif`. Если проверяемое условие справедливо, то строки между `#else` и `#endif` игнорируются. Если же проверяемое условие не выполняется, то игнорируются все строки между проверкой и командой `#else`, а если ее нет, то командой `#endif`.

**Пример:**

```
#ifndef TestMode
#define TestMode
#endif
//+-----+
|| Script program start function |
//+-----+
void OnStart()
{
    #ifdef TestMode
        Print("Test mode");
    #else
        Print("Normal mode");
    #endif
}
```

В зависимости от типа программы и режима компиляции стандартные макросы определяются следующим образом:

Макрос `_MQL5_` доступен при компиляции файла `*.mq5`, при компиляции `*.mq4` доступен макрос `_MQL4_`.

Макрос `_DEBUG` доступен при компиляции под отладку.

Макрос `_RELEASE` доступен при компиляции не под отладку.

**Пример:**

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    #ifdef __MQL5__
        #ifdef _DEBUG
            Print("Hello from MQL5 compiler [DEBUG]");
        #else
            #ifdef _RELEASE
                Print("Hello from MQL5 compiler [RELEASE]");
            #endif
        #endif
    #else
        #ifdef __MQL4__
            #ifdef _DEBUG
                Print("Hello from MQL4 compiler [DEBUG]");
            #else
                #ifdef _RELEASE
                    Print("Hello from MQL4 compiler [RELEASE]");
                #endif
            #endif
        #endif
    #endif
}
```

## Объектно-ориентированное программирование

Объектно-ориентированное программирование - это программирование, сфокусированное на данных, причем данные и поведение неразрывно связаны между собой. Вместе данные и поведение представляют собой класс, а объекты являются экземплярами класса.

Составными частями объектно-ориентированного подхода являются:

- [Инкапсуляция и расширяемость типов](#)
- [Наследование](#)
- [Полиморфизм](#)
- [Перегрузка](#)
- [Виртуальные функции](#)

ООП рассматривает вычисления как моделирование поведения. То, что моделируется, является объектами, представленными вычислительными абстракциями. Допустим, мы хотим написать хорошо всем известную игру "Тетрис", для этого мы должны научиться моделировать появление случайной фигуры, составленной из четырех квадратиков, соединенных друг с другом ребрами. Также требуется регулировать скорость падения фигуры, задать операции вращения и сдвига фигуры. Перемещения фигуры на экране ограничены границами стакана, это требование мы также должны смоделировать. Кроме того, заполненные ряды кубиков в стакане должны уничтожаться и необходимо вести подсчет очков, заработанных в игре.

Таким образом, такая простая в понимании игра требует создания нескольких моделей - модель фигуры, модель стакана, модель движения фигуры в стакане и так далее. Все эти модели являются абстракциями, представленными вычислениями в компьютере. Для описания таких моделей применяют понятие *абстрактный тип данных*, АТД (или [сложный тип данных](#)). Строго говоря, модель движения "фигуры" в "стакане" не является типом данных, а является совокупностью операций над данными типа "фигура", использующих ограничения данных типа "стакан".

Объекты являются переменными [класса](#). Объектно-ориентированное программирование позволяет легко создавать и использовать АТД. Объектно-ориентированное программирование использует механизм [наследования](#). Наследование выгодно тем, что позволяет получать производные типы из уже определенных пользователем типов данных. Так, для создания фигур в тетрисе удобно сначала создать базовый класс Shape, на основе которого получены производные типы всех семи возможных в тетрисе фигур. В базовом классе определено поведение фигур, а в производных уточнена реализация поведения каждой конкретной фигуры.

В ООП объекты отвечают за свое поведение. Разработчик АТД должен включать в него код для описания любого поведения, которое обычно можно ожидать от соответствующих объектов. То, что объект сам отвечает за свое поведение, значительно упрощает задачу программирования для пользователя этого объекта.

Если мы хотим нарисовать на экране фигуру, нам надо знать где будет находиться ее центр и как ее рисовать. Если отдельная фигура прекрасно понимает, как себя нарисовать, программист при использовании такой фигуры должен лишь передать объекту сообщение "нарисоваться".

Язык MQL5 является C++ подобным, и в нем также реализован механизм [инкапсуляции](#) для реализации АТД. Инкапсуляция сочетает в себе, с одной стороны, внутренние детали реализации конкретного типа и, с другой, доступные извне функции, которые могут действовать на объекты

этого типа. Детали реализации могут быть недоступны для программы, которая использует данный тип.

К понятию ООП имеет отношение целый набор концепций, включая нижеследующие:

- Моделирование действий из реального мира
- Наличие типов данных, определяемых пользователем
- Сокрытие деталей реализации
- Возможность многократного использования кода благодаря наследованию
- Интерпретация вызовов функций на этапе исполнения

Некоторые из этих понятий довольно расплывчаты, некоторые - абстрактны, другие носят общий характер.

## Инкапсуляция и расширяемость типов

ООП - это сбалансированный подход к написанию программного обеспечения. Данные и поведение упакованы вместе. Такая инкапсуляция создает определяемые пользователем типы данных, расширяющие собственные типы данных языка и взаимодействующие с ними. Расширяемость типов - это возможность добавлять к языку определяемые пользователем типы данных, которые также легко использовать, как и [основные типы](#).

Абстрактный тип данных, например, строка, является описанием идеального, всем известного поведения типа. Пользователь строки знает, что операции, такие как конкатенация или печать, имеют определенное поведение. Операции конкатенации и печати называются методами.

Конкретная реализация АТД может иметь ограничения; например, строки могут быть ограничены по длине. Эти ограничения влияют на открытое всем поведение. В то же время, внутренние или закрытые детали реализации не влияют прямо на то, как пользователь видит объект. Например, строка часто реализуется как массив; при этом внутренний базовый адрес элементов этого массива и его имя не существенны для пользователя.

Инкапсуляция - это способность скрывать внутренние детали при предоставлении открытого интерфейса к определяемому пользователем типу. В MQL5, как и в C++, для обеспечения инкапсуляции используются определения класса и структуры ([class](#) и [struct](#)) в сочетании с ключевыми словами доступа [private](#) (закрытый), [protected](#) (защищенный) и [public](#) (открытый).

Ключевое слово [public](#) показывает, что доступ к членам, которые стоят за ним, является открытым без всяких ограничений. Без этого ключевого слова члены класса по умолчанию закрыты. Закрытые члены доступны только функциям-членам только своего класса.

Защищенные члены класса доступны функциям-членам не только своего класса, но и классов-наследников. Открытые члены доступны любой функции внутри области видимости объявления класса. Закрытость позволяет спрятать часть реализации класса, предотвращая тем самым непредвиденные изменения структуры данных. Ограничение доступа или скрытие данных является особенностью объектно-ориентированного программирования.

Обычно стараются защищать члены класса и объявлять их с модификатором [protected](#), установку и чтение значений этих членов осуществляется с помощью специальных так называемых [set](#)- и [get](#)-методов, которые определяются с модификатором доступа [public](#).

**Пример:**

```
class CPerson
{
protected:
    string m_name; // имя
public:
    void SetName(string n) {m_name=n;} // устанавливает имя
    string GetName() {return (m_name);} // возвращает имя
};
```

Такой подход дает несколько преимуществ. Во-первых, по имени функции можно понять что она делает - устанавливает или получает значение члена класса. Во-вторых, возможно в будущем нам понадобится изменить тип переменной `m_name` в самом классе `CPerson` или в каком-либо из производных от него классов.

В таком случае нам достаточно будет изменить реализацию функций SetName() и GetName(), сами же объекты класса CPerso~~n~~ можно будет использовать в программе без каких-либо изменений в коде, так как пользователь не будет даже знать, что тип данных m\_name изменился.

**Пример:**

```

struct Name
{
    string first_name; // имя
    string last_name; // фамилия
};

class CPerson
{
protected:
    Name m_name; // имя
public:
    void SetName(string n);
    string GetName() { return (m_name.first_name+" "+m_name.last_name); }
private:
    string GetFirstName(string full_name);
    string GetLastName(string full_name);
};

void CPerson::SetName(string n)
{
    m_name.first_name=GetFirstName(n);
    m_name.last_name=GetLastName(n);
}

string CPerson::GetFirstName(string full_name)
{
    int pos=StringFind(full_name, " ");
    if (pos>0)
        StringSetCharacter(full_name, pos, 0);
    return (full_name);
}

string CPerson::GetLastName(string full_name)
{
    string ret_string;
    int pos=StringFind(full_name, " ");
    if (pos>0)
        ret_string=StringSubstr(full_name, pos+1);
    else
        ret_string=full_name;
//---
    return (ret_string);
}

```

Смотри также

[Типы данных](#)

## Наследование

Особенностью ООП является поощрение повторного использования кода при помощи механизма наследования. Новый класс производится от существующего, называемого базовым классом. Производный класс использует члены базового класса, но может также изменять и дополнять их.

Многие типы представляют собой вариации на темы существующих. Часто бывает утомительно разрабатывать новый код для каждого из них. Кроме того, новый код - новые ошибки. Производный класс наследует описание базового класса, делая ненужными повторную разработку и тестирование кода. Отношения наследования иерархичны.

Иерархия - это метод, позволяющий копировать элементы во всем их многообразии и сложности. Она вводит классификацию объектов. Например, в периодической системе элементов есть газы. Они обладают свойствами, присущими всем элементам системы.

Инертные газы - следующий важный подкласс. Иерархия заключается в том, что инертный газ, например, аргон - это газ, а газ, в свою очередь является элементом системы. Такая иерархия позволяет легко толковать поведение инертных газов. Мы знаем, что их атомы содержат протоны и электроны, что верно и для прочих элементов.

Мы знаем, что они пребывают в газообразном состоянии при комнатной температуре, как все газы. Мы знаем, что ни один газ из подкласса инертных газов не вступает в обычную химическую реакцию ни с одним из элементов, и это свойство всех инертных газов.

Рассмотрим наследование на примере геометрических фигур. Для описания всего многообразия простых фигур (круг, треугольник, прямоугольник, квадрат и так далее) лучше всего создать базовый класс ([АТД](#)), который является прародителем всех производных классов.

Создадим базовый класс `CShape`, в котором есть только самые общие члены, описывающие фигуру. Эти члены описывают свойства, присущие любой фигуре - тип фигуры и координаты основной точки привязки.

**Пример:**

```
//--- Базовый класс Фигура
class CShape
{
protected:
    int      m_type;           // тип фигуры
    int      m_xpos;          // X - координата точки привязки
    int      m_ypos;          // Y - координата точки привязки
public:
    CShape() {m_type=0; m_xpos=0; m_ypos=0;} // конструктор
    void    SetXPos(int x) {m_xpos=x;} // установим X
    void    SetYPos(int y) {m_ypos=y;} // установим Y
};
```

Далее создадим от базового класса производные классы, в которых добавим необходимые поля, уточняющие каждый конкретный класс. Для фигуры `Circle` (круг) необходимо добавить член, который содержит значение радиуса. Фигура `Quadrat` (квадрат) характеризуется значением стороны квадрата. Поэтому производные классы, унаследованные от базового класса `CShape`, будут объявлены таким образом:

```
//--- производный класс Круг
class CCircle : public CShape // после двоеточия указывается базовый класс,
{ // от которого производится наследование
private:
    int m_radius; // радиус круга

public:
    CCircle(){m_type=1;} // конструктор, тип равен 1
};
```

Для квадрата объявление класса выглядит аналогично:

```
//--- производный класс Квадрат
class CSquare : public CShape // после двоеточия указывается базовый класс,
{ // от которого производится наследование
private:
    int m_square_side; // сторона квадрата

public:
    CSquare(){m_type=2;} // конструктор, тип равен 2
};
```

Необходимо отметить, что при создании объекта сначала вызывается конструктор базового класса, затем [конструктор](#) производного класса. При уничтожении объекта сначала вызывается [деструктор](#) производного класса, а затем деструктор базового класса.

Таким образом, объявив в базовом классе наиболее общие члены, в производных классах мы можем добавлять дополнительные члены, которые уточняют конкретный класс. Наследование позволяет создавать мощные библиотеки кода, которые можно использовать многократно и повторно.

Синтаксис создания производного класса от уже существующего выглядит следующим образом:

```
class имя_класса :
    (public | protected | private) opt имя_базового_класса
{
    объявления членов
};
```

Одним из аспектов производного класса является видимость (открытость) его членов-наследников. Ключевые слова `public`, `protected` и `private` используются для указания того, насколько члены базового класса будут доступны из производного. Использование в заголовке производного класса ключевого класса `public`, следующего за двоеточием, означает, что защищенные и открытые (`protected` и `public`) члены базового класса `CShape` должны наследоваться как защищенные и открытые члены производного класса `CCircle`.

Закрытые члены базового класса недоступны производному классу. Открытое наследование означает также, что производные классы (`CCircle` и `CSquare`) являются `CShape`. То есть, квадрат (`CSquare`) является фигурой (`CShape`), но фигура не обязательно должна быть квадратом.

Производный класс является модификацией базового класса; он наследует защищенные и открытые члены базового класса. Не могут только наследоваться конструкторы и деструкторы

базового класса. Часто в производный класс добавляются новые члены в дополнение к членам базового класса.

Производный класс может включать реализацию функций-членов, отличную от базового класса. Это не имеет ничего общего с [перегрузкой](#), когда смысл одного и того же имени функции может быть различным для разных сигнатур.

При защищенном наследовании открытые и защищенные члены базового класса становятся защищенными членами производного класса. При закрытом наследовании открытые и защищенные члены базового класса становятся закрытыми членами производного класса.

При защищенном и закрытом наследовании не справедливо отношение, что объект производного класса является объектом базового класса. Защищенное и закрытое наследование встречаются редко и каждое из них нужно использовать с большой осторожностью.

Необходимо понимать, что тип наследования (public, protected или private) никак не влияет на способы доступа к членам базовых классов в иерархии наследования из потомка (наследника). При любом типе наследования из производных классов будут доступны только члены базового класса, объявленные со спецификаторами доступа public и protected. Рассмотрим вышеизложенное на примере:

```
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

//+-----+
// | Класс-пример с несколькими типами доступа           |
//+-----+
class CBaseClass
{
    private:           //--- закрытый член недоступен из потомков
        int           m_member;
    protected:         //--- защищенный метод доступен из базового класса и его потомков
        int           Member() {return (m_member);}
    public:            //--- конструктор класса доступен всем
        CBaseClass() {m_member=5;return;};
    private:           //--- закрытый метод для присвоения значения члену m_member
        void          Member(int value) { m_member=value;};

    };
//+-----+
// | Производный класс с ошибками                         |
//+-----+
class CDerived: public CBaseClass // public наследование можно не указывать, оно по умолчанию
{
    public:
        void Func() // определим в потомке функцию с обращениями к членам базового класса
        {
            //--- попытка модификации закрытого члена базового класса
            m_member=0;           // ошибка, закрытый член базового класса никому не доступен
            Member(0);           // ошибка, закрытый метод базового класса не доступен в потомке
        }
}
```

```

    //--- чтение члена базового класса
    Print(m_member);      // ошибка, закрытый член базового класса никому не доступен
    Print(Member());      // нет ошибки, защищенный метод доступен из базового класса
}
};

```

В приведенном примере класс CBaseClass имеет только один публичный метод - конструктор. Конструкторы вызываются автоматически при создании объекта класса. Поэтому извне никак нельзя обратиться ни к закрытому члену m\_member, ни к защищенному методу Member(). Но при этом при открытом (public) наследовании метод Member() базового класса будет доступен из потомков.

При защищенном (protected) наследовании все члены базового класса с открытым и защищенным доступом становятся защищенными. Это означает, что если открытые члены-данные и методы базового класса были доступны извне, то при защищенном наследовании теперь они доступны только из классов самого потомка и его последующих производных классах.

```

//+-----+
// | Класс-пример с несколькими типами доступа
//+-----+
class CBaseMathClass
{
private:           //--- закрытый член недоступен из потомков
    double m_Pi;
public:            //--- получение и установка значения для m_Pi
    void SetPI(double v){m_Pi=v;return;};
    double GetPI(){return m_Pi;};
public:            // конструктор класса доступен всем
    CBaseMathClass() {SetPI(3.14); PrintFormat("%s", __FUNCTION__);}
};

//+-----+
// | Производный класс, в котором нельзя уже изменить m_Pi
//+-----+
class CProtectedChildClass: protected CBaseMathClass // защищенное наследование
{
private:
    double m_radius;
public:             //--- открытые методы в потомке
    void SetRadius(double r){m_radius=r; return;};
    double GetCircleLength(){return GetPI()*m_radius;};
};

//+-----+
// | Функция запуска скрипта
//+-----+
void OnStart()
{
//--- при создании потомка конструктор базового класса вызовется автоматически
    CProtectedChildClass pt;
//--- укажем радиус
    pt.SetRadius(10);
}

```

```
PrintFormat("Length=%G",pt.GetCircleLength());  
//--- если раскомментировать строку ниже, то получим ошибку на этапе компиляции, так как  
// pt.SetPI(3);  
  
//--- а теперь объявим переменную базового класса и попробуем задать константу Pi равной  
CBaseMathClass bc;  
bc.SetPI(10);  
//--- посмотрим что получилось  
PrintFormat("bc.GetPI()=%G",bc.GetPI());  
}
```

В данном пример показано, что методы `SetPI()` и `GetPi()` в базовом классе `CBaseMathClass` являются открытыми и доступны для вызова из любого места программы. Но в то же время для его потомка `CProtectedChildClass` вызовы этих методов можно делать только из методов самого класса `CProtectedChildClass` или его потомков.

При закрытом (`private`) наследовании все члены базового класса с доступом `public` и `protected` становятся закрытыми, и при дальнейшем наследовании обращение к ним становится невозможным.

В MQL5 нет множественного наследования.

#### Смотри также

[Структуры и классы](#)

## Полиморфизм

Полиморфизм - это возможность для объектов разных классов, связанных с помощью наследования, реагировать различным способом при обращении к одной и той же функции-элементу. Это помогает создавать универсальные механизмы, описывающие поведение не только базового класса, но и классов-потомков.

Продолжим разработку базового класса CShape, в котором определим функцию-член GetArea(), предназначенную для расчета площади фигуры. Во всех классах-потомках, произведенных наследованием от базового класса, мы переопределим эту функцию в соответствие с правилами расчета площади конкретной фигуры.

Для квадрата (класс CSquare) площадь вычисляется через стороны, для круга (класс CCircle) площадь выражается через радиус и так далее. Мы можем создать массив для хранения объектов типа CShape, в котором можно будет хранить как объект базового класса, так и всех его потомков. В дальнейшем мы можем вызывать одну и ту же функцию для любого элемента данного массива.

**Пример:**

```
//--- Базовый класс
class CShape
{
protected:
    int          m_type;           // тип фигуры
    int          m_xpos;          // X - координата точки привязки
    int          m_ypos;          // Y - координата точки привязки
public:
    void        CShape() {m_type=0;} // конструктор, тип равен нулю
    int         GetType() {return(m_type);} // возвращает тип фигуры
virtual
    double      GetArea() {return (0); } // возвращает площадь фигуры
};
```

Теперь все производные классы имеют функцию-член getArea(), которая возвращает нулевое значение. Реализация этой функции в каждом потомке будет отличаться.

```
//--- производный класс Круг
class CCircle : public CShape           // после двоеточия указывается базовый класс,
{                                         // от которого производится наследование
private:
    double      m_radius;            // радиус круга

public:
    void        CCircle() {m_type=1;} // конструктор, тип равен 1
    void        SetRadius(double r) {m_radius=r;};
    virtual double GetArea() {return (3.14*m_radius*m_radius);} // площадь круга
};
```

Для квадрата объявление класса выглядит аналогично:

```
//--- производный класс Квадрат
```

```

class CSquare : public CShape           // после двоеточия указывается базовый класс,
{                                      // от которого производится наследование
private:
    double      m_square_side;        // сторона квадрата

public:
    void        CSquare() {m_type=2;} // конструктор, тип равен 2
    void        SetSide(double s) {m_square_side=s;};
    virtual double GetArea() {return (m_square_side*m_square_side);} //площадь квадрата
};

```

Так как для вычисления площади квадрата и круга требуются соответствующие значения членов `m_radius` и `m_square_side`, то в объявлении соответствующего класса мы добавили функции `SetRadius()` и `SetSide()`.

Предполагается, что в программе у нас используются объекты разного типа (`CCircle` и `CSquare`), но унаследованные от одного базового типа `CShape`. Полиморфизм позволяет нам создать массив объектов базового типа `CShape`, но при объявлении этого массива сами объекты еще неизвестны и их тип неопределен.

Решение о том, объект какого типа будет содержаться в каждом элементе массива, будет приниматься непосредственно при выполнении программы. Это подразумевает [динамическое создание](#) объектов соответствующих классов, и, следовательно, необходимость вместо самих объектов использовать [указатели объектов](#).

Для динамического создания объектов используется оператор `new`, каждый такой объект нужно самостоятельно и явно удалять оператором `delete`. Поэтому мы объявим массив указателей типа `CShape` и для каждого его элемента создадим объект нужного типа (`new Имя_класса`), как это показано в примере скрипта:

```

//+-----+
//| Script program start function          |
//+-----+
void OnStart()
{
//--- объявили массив указателей объектов базового типа
CShape *shapes[5]; // массив указателей на объекты CShape

//--- здесь заполняем массив производными объектами
//--- объявили указатель на объект типа CCircle
CCircle *circle=new CCircle();
//--- задаем свойства объекта по указателю circle
circle.SetRadius(2.5);
//--- поместим в shapes[0] значение указателя
shapes[0]=circle;

//--- создаем еще один объект CCircle и запишем его указатель в shapes[1]
circle=new CCircle();
shapes[1]=circle;
circle.SetRadius(5);

```

```

//--- тут мы намеренно "забыли" задать значение для shapes[2]
//circle=new CCircle();
//circle.SetRadius(10);
//shapes[2]=circle;

//--- для неиспользуемого элемента установим значение NULL
shapes[2]=NULL;

//--- создаем объект CSquare и запишем его указатель в shapes[3]
CSquare *square=new CSquare();
square.SetSide(5);
shapes[3]=square;

//--- создаем объект CSquare и запишем его указатель в shapes[4]
square=new CSquare();
square.SetSide(10);
shapes[4]=square;

//--- массив указателей есть, получим его размер
int total=ArraySize(shapes);

//--- пройдем в цикле по всем указателям в массиве
for(int i=0; i<5;i++)
{
    //--- если по указанному индексу указатель является валидным
    if(CheckPointer(shapes[i])!=POINTER_INVALID)
    {
        //--- выведем в лог тип и площадь фигуры
        PrintFormat("Объект типа %d имеет площадь %G",
            shapes[i].GetType(),
            shapes[i].GetArea());
    }
    //--- если указатель имеет тип POINTER_INVALID
    else
    {
        //--- сообщим об ошибке
        PrintFormat("Объект shapes[%d] не инициализирован! Его указатель %s",
            i,EnumToString(CheckPointer(shapes[i])));
    }
}

//--- мы должны самостоятельно уничтожить все созданные динамические объекты
for(int i=0;i<total;i++)
{
    //--- удалять можно только объекты, чей указатель имеет тип POINTER_DYNAMIC
    if(CheckPointer(shapes[i])==POINTER_DYNAMIC)
    {
        //--- сообщим об удалении
        PrintFormat("Удаляем shapes[%d]",i);
        //--- уничтожим объект по его указателю
    }
}

```

```
    delete shapes[i];
}
}
}
```

Обратите внимание, что при уничтожении объекта оператором `delete` необходимо проверить [типа его указателя](#). Удалять с помощью `delete` можно только объекты, имеющие указатель [POINTER\\_DYNAMIC](#), для указателей другого типа будет получена ошибка.

Кроме переопределения функции при наследовании, полиморфизм включает в себя также и реализацию одной и той же функции с разным набором параметров в пределах одного класса. Это означает, что в классе может быть определено несколько функций с одним и тем же именем, но с разным типом и/или набором параметров. В этом случае полиморфизм реализуется через [перегрузку функций](#).

**Смотри также**

[Стандартная библиотека](#)

## Перегрузка

В пределах одного класса можно определить два или более методов, которые совместно используют одно и тоже имя, но имеют разное количество параметров. Когда это имеет место, методы называются *перегруженными*, а о процессе говорят как о *перегрузке метода*. Перегрузка метода - один из способов, с помощью которых реализуется [полиморфизм](#). Перегрузка методов в классах производится по тем же правилам, что и [перегрузка функций](#).

Если для вызываемой функции нет точного соответствия, то компилятор производит поиск подходящей функции по трем уровням последовательно:

1. поиск среди методов класса;
2. поиск среди методов базовых классов, последовательно от ближайшего предка до самого первого;
3. поиск среди остальных функций.

Если точного соответствия ни на одном уровне не найдено, но найдено несколько подходящих функций на разных уровнях, то используется функция, найденная на наименьшем уровне. В пределах одного уровня не может быть более одной подходящей функции.

Смотри также

[Перегрузка функций](#)

## Виртуальные функции

Ключевое слово `virtual` служит спецификатором функции, который обеспечивает механизм для динамического выбора на этапе выполнения подходящей функции-члена среди функций базового и производного классов, структуры не могут иметь виртуальных функций. Оно может применяться для изменения [объявлений](#) только функций-членов.

Виртуальная функция, как и обычная функция, должна иметь [исполняемое тело](#). При вызове семантика ее точно такая же, как и у остальных функций.

Виртуальная функция может замещаться в производном классе. Выбор того, какое [определение функции](#) вызвать для виртуальной функции, происходит динамически (на этапе выполнения). Типичный случай - когда базовый класс содержит виртуальную функцию, а производные классы имеют свои версии этой функции.

Указатель на базовый класс может указывать либо на объект базового класса, либо на объект производного класса. Выбор вызываемой функции-члена будет произведен на этапе выполнения и будет зависеть от типа объекта, а не от типа указателя. При отсутствии члена производного типа по умолчанию используется виртуальная функция базового класса.

[Деструкторы](#) всегда являются виртуальными, независимо от того, объявлены они с ключевым слово `virtual` или нет.

Рассмотрим использование виртуальных функций на примере программы `MT5_Tetris.mq5`. Во включаемом файле `MT5_TetrisShape.mqh` определен базовый класс `CTetrisShape` с виртуальной функцией `Draw` (рисовать).

```
//+-----+
class CTetrisShape
{
protected:
    int          m_type;
    int          m_xpos;
    int          m_ypos;
    int          m_xsize;
    int          m_ysize;
    int          m_prev_turn;
    int          m_turn;
    int          m_right_border;
public:
    void         CTetrisShape();
    void         SetRightBorder(int border) { m_right_border=border; }
    void         SetYPos(int ypos)           { m_ypos=ypos; }
    void         SetXPos(int xpos)          { m_xpos=xpos; }
    int          GetYPos()                { return(m_ypos); }
    int          GetXPos()                { return(m_xpos); }
    int          GetYSize()               { return(m_ysize); }
    int          GetXSize()               { return(m_xsize); }
    int          GetType()                { return(m_type); }
    void         Left()                  { m_xpos-=SHAPE_SIZE; }
    void         Right()                 { m_xpos+=SHAPE_SIZE; }
```

```

void           Rotate()          { m_prev_turn=m_turn; if(++m_turn>3) r
virtual void   Draw()           { return; }
virtual bool   CheckDown(int& pad_array[]);
virtual bool   CheckLeft(int& side_row[]);
virtual bool   CheckRight(int& side_row[]);
};

}

```

Далее для каждого производного класса эта функция реализована в соответствии с особенностями класса-потомка. Например, первая фигура CTetrisShape1 имеет свою реализацию функции Draw():

```

class CTetrisShape1 : public CTetrisShape
{
public:
    //--- отрисовка фигуры
    virtual void     Draw()
    {
        int      i;
        string  name;
        //---
        if(m_turn==0 || m_turn==2)
        {
            //--- горизонтальная палка
            for(i=0; i<4; i++)
            {
                name=SHAPE_NAME+(string)i;
                ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+i*SHAPE_SIZE);
                ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos);
            }
        }
        else
        {
            //--- вертикальная палка
            for(i=0; i<4; i++)
            {
                name=SHAPE_NAME+(string)i;
                ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos);
                ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos+i*SHAPE_SIZE);
            }
        }
    }
}

```

Фигура квадрат описана классом CTetrisShape6 и имеет собственную реализацию метода Draw():

```

class CTetrisShape6 : public CTetrisShape
{
public:
    //--- отрисовка фигуры
    virtual void     Draw()
    {

```

```

int      i;
string  name;
//---
for(i=0; i<2; i++)
{
    name=SHAPE_NAME+(string)i;
    ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpost+i*SHAPE_SIZE);
    ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos);
}
for(i=2; i<4; i++)
{
    name=SHAPE_NAME+(string)i;
    ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpost+(i-2)*SHAPE_SIZE);
    ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos+SHAPE_SIZE);
}
}
};

```

В зависимости от того, объект какого класса создан, вызывается виртуальная функция того или иного производного класса.

```

void CTetrisField::NewShape()
{
//--- случайным образом создаём одну из 7 возможных фигур
int nshape=rand()%7;
switch(nshape)
{
    case 0: m_shape=new CTetrisShape1; break;
    case 1: m_shape=new CTetrisShape2; break;
    case 2: m_shape=new CTetrisShape3; break;
    case 3: m_shape=new CTetrisShape4; break;
    case 4: m_shape=new CTetrisShape5; break;
    case 5: m_shape=new CTetrisShape6; break;
    case 6: m_shape=new CTetrisShape7; break;
}
//--- отрисовываем
m_shape.Draw();
//---
}

```

## Модификатор `override`

Модификатор `override` означает, что объявляемая функция обязательно должна переопределить метод родительского класса. Использование этого модификатора позволяет избежать ошибок при переопределении, таких как случайное изменение сигнатуры метода. Например, в базовом классе определен метод `func`, принимающий в качестве аргумента переменную типа `int`:

```

class CFoo
{
    void virtual func(int x) const { }
}

```

```
};
```

Далее метод переопределяется в наследуемом классе:

```
class CBar : public CFoo
{
    void func(short x) { }
};
```

Но по ошибке тип аргумента изменяется с `int` на `short`. Фактически, в этом случае уже происходит не переопределение, а перегрузка метода. Действуя в соответствии с [алгоритмом определения перегруженной функции](#), в определенных ситуациях компилятор может выбрать метод, определенный в базовом классе, вместо переопределенного метода.

Чтобы избежать подобных ошибок, к переопределяемому методу следует явно добавлять модификатор `override`.

```
class CBar : public CFoo
{
    void func(short x) override { }
};
```

Если при переопределении будет изменена сигнатура метода, компилятор не сможет найти в родительском классе метод с точной такой же сигнатурой и выдаст ошибку компиляции:

```
'CBar::func' method is declared with 'override' specifier but does not override any ba
```

## Модификатор `final`

Модификатор `final` действует наоборот — он запрещает переопределение метода в классах-наследниках. Если реализация метода самодостаточна и полностью завершена, объявит его с модификатором `final`, чтобы он гарантированно не был изменен в последующем.

```
class CFoo
{
    void virtual func(int x) final { }
};

class CBar : public CFoo
{
    void func(int) { }
};
```

При попытке переопределения метода с модификатором `final`, как показано в примере выше, компилятор выдаст ошибку:

```
'CFoo::func' method declared as 'final' cannot be overridden by 'CBar::func'
see declaration of 'CFoo::func'
```

**Смотри также**

[Стандартная библиотека](#)

## Статические члены класса/структурь

### Статические члены

Члены класса могут быть объявлены с использованием модификатора класса памяти [static](#). Такие члены данных разделяются всеми экземплярами данного класса и хранятся в одном месте. Нестатические члены данных создаются для каждой переменной-объекта класса.

Отсутствие возможности объявлять статически члены класса привело бы к необходимости объявлять эти данные на [глобальном уровне](#) программы. Это разорвало бы отношения между данными и их классом, а также не согласуется с основной парадигмой ООП - объединение в классе данных и методов для их обработки. Статический член позволяет данным класса, которые не специфичны для отдельного экземпляра, существовать в области видимости класса.

Так как статический член класса не зависит от конкретного экземпляра, то обращение к нему выглядит следующим образом:

```
class_name::variable
```

где *class\_name* - это имя класса, а *variable* означает имя члена класса.

Как видите, для обращения к статическому члену класса используется [оператор разрешения контекста ::](#). При обращении к статическому члену внутри методов класса оператор контекста необязателен.

Статический член класса требуется явно инициализировать нужным значением, для этого он должен быть объявлен и проинициализирован на глобальном уровне. Порядок инициализации статических членов будет соответствовать порядку их объявления на глобальном уровне в исходном коде.

Например, у нас есть класс *CParser*, предназначенный для синтаксического разбора текстов, и нам необходимо считать общее количество обработанных слов и символов. Достаточно объявить нужные члены класса статическими и инициализировать их на глобальном уровне. Тогда все экземпляры класса при работе будут использовать общие счетчики слов и символов.

```
//+-----+
// | Класс "Анализатор текстов"
// +-----+
class CParser
{
public:
    static int      s_words;
    static int      s_symbols;
    //--- конструктор и деструктор
    CParser(void);
    ~CParser(void) {};
};

...
//--- инициализация статических членов класса Parser на глобальном уровне
int CParser::s_words=0;
int CParser::s_symbols=0;
```

Статический член класса можно объявить с ключевым словом `const`. Такие статические константы должны быть инициализированы на глобальном уровне с ключевым словом `const`:

```
//+-----+
// | Класс "Стек" для хранения обрабатываемых данных |
//+-----+
class CStack
{
public:
    CStack(void);
    ~CStack(void) {};
...
private:
    static const int s_max_length; // максимальная емкость стека
};

//--- инициализация статической константы класса CStack
const int CStack::s_max_length=1000;
```

## Указатель `this`

Ключевое слово `this` обозначает объявленный неявно указатель на себя - на конкретный экземпляр класса, в контексте которого выполняется метод. Он может использоваться только в нестатических методах класса. Указатель `this` является неявным нестатическим членом любого класса.

В статических функциях можно обращаться только к статическим членам/методам класса.

## Статические методы

В MQL5 разрешается использовать функции-члены типа `static`. Модификатор `static` должен идти перед возвращаемым типом функции в объявлении внутри класса.

```
class CStack
{
public:
    //--- конструктор и деструктор
    CStack(void) {};
    ~CStack(void) {};

    //--- максимальная емкость стека
    static int Capacity();

private:
    int m_length;      // количество элементов в стеке
    static const int s_max_length; // максимальная емкость стека
};

//+-----+
// | Возвращает максимальную вместимость стека |
//+-----+
int CStack::Capacity(void)
{
```

```

        return(s_max_length);
    };
//--- инициализация статической константы класса CStack
const int CStack::s_max_length=1000;
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- объявим переменную типа CStack
CStack stack;
//--- вызываем статический метод объекта
Print("CStack.s_max_length=",stack.Capacity());
//--- а можно и так вызвать, т.к. метод статический и не требует наличия объекта
Print("CStack.s_max_length=",CStack::Capacity());
}

```

Метод с модификатором **const** называется **постоянным** и не может модифицировать неявные члены своего класса. Объявление постоянных функций класса и постоянных параметров называется **контролем постоянства** (*const-correctness*). Благодаря такому контролю можно быть уверенными, что компилятор проследит за неизменностью значений объектов и выдаст ошибку еще на стадии компиляции в случае нарушения.

Модификатор **const** ставится после списка аргументов внутри объявления класса. Определение вне класса также должно включать модификатор **const**:

```

//+-----+
//| Класс "Прямоугольник"
//+-----+
class CRectangle
{
private:
    double          m_width;      // ширина
    double          m_height;     // высота
public:
    //--- конструкторы и деструктор
    CRectangle(void) :m_width(0),m_height(0){};
    CRectangle(const double w,const double h) :m_width(w),m_height(h) ·
    ~CRectangle(void){};

    //--- вычисление площади
    double          Square(void) const;
    static double   Square(const double w,const double h); // { return(w*h); }

};

//+-----+
//| Возвращает площадь объекта "Прямоугольник"
//+-----+
double CRectangle::Square(void) const
{
    return(Square(m_width,m_height));
}

```

```
//+-----+
//| Возвращает произведение двух переменных |
//+-----+
static double CRectangle::Square(const double w,const double h)
{
    return (w*h);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- создадим прямоугольник rect со сторонами 5 и 6
    CRectangle rect(5,6);
//--- выведем площадь прямоугольника константным методом
    PrintFormat("rect.Square()=%.2f",rect.Square());
//--- выведем произведение чисел с помощью статического метода класса CRectangle
    PrintFormat("CRectangle::Square(2.0,1.5)=%f",CRectangle::Square(2.0,1.5));
}
```

Дополнительным аргументом в пользу использования контроля целостности служит то, что компилятор в этом случае производит специальную оптимизацию, например, располагает постоянный объект в памяти только для чтения.

Статическая функция не может быть определена с модификатором `const`, так как данный модификатор гарантирует неизменность членов экземпляра при вызове такой функции. Но, как уже говорилось выше, статическая функция по определению не может обращаться к нестатическим членам класса.

#### Смотри также

[Статические переменные](#), [Переменные](#), [Ссылки](#). [Модификатор & и ключевое слово this](#)

## Шаблоны функций

Перегруженные функции обычно используются для выполнения похожих операций над различными типами данных. Простой пример такой функции в MQL5 - [ArraySize\(\)](#), которая возвращает размер массива любого типа. На самом деле эта системная функция является перегруженной, и вся реализация такой перегрузки спрятана от разработчика программ на MQL5:

```
int ArraySize(
    void& array[] // проверяемый массив
);
```

То есть на самом деле компилятор языка MQL5 подставляет для каждого вызова этой функции нужную реализацию, например, для массивов целого типа примерно так:

```
int ArraySize(
    int& array[] // массив с элементами типа int
);
```

А для массива типа [MqlRates](#) для работы с котировками в формате исторических данных функцию [ArraySize\(\)](#) можно представить таким образом:

```
int ArraySize(
    MqlRates& array[] // массив, заполненный значениями типа MqlRates
);
```

Таким образом, очень удобно использовать одну и ту же функцию для работы с разными типами, но необходимо самостоятельно провести всю предварительную работу, а именно - сделать перегрузку нужной функции для всех типов данных, с которыми она должна будет корректно работать.

Есть более удобное решение - если для каждого типа данных должны выполняться идентичные операции, то более компактным и удобным решением является использование шаблонов функций. При этом программисту нужно написать всего одно описание шаблона функции. При таком описании шаблона в качестве параметра достаточно указать не конкретный тип данных, с которыми должна работать функция, а некий формальный параметр. Основываясь на типах аргументов, использованных при вызове этой функции, компилятор будет автоматически генерировать разные функции для соответствующей обработки каждого типа.

Определение шаблона функции начинается с ключевого слова [template](#), после которого в угловых скобках идет список формальных параметров. Каждый формальный параметр предваряется ключевым словом [typename](#). Формальные типы параметров - встроенные типы или типы, определенные пользователем. Они используются:

- для задания типов аргументов функции,
- для задания типов возвращаемого значения функции,
- для объявления переменных внутри тела описания функции

Количество параметров в шаблоне не может быть больше восьми. Каждый формальный параметр в определении шаблона должен хотя бы один раз появиться в списке параметров функции. Каждое имя формального параметра должно быть уникальным.

Вот пример шаблона функции для поиска максимального значения в массиве любого числового типа (целые и вещественные числа):

```
template<typename T>
T ArrayMax(T &arr[])
{
    uint size=ArraySize(arr);
    if(size==0)
        return(0);

    T max=arr[0];
    for(uint n=1;n<size;n++)
        if(max<arr[n]) max=arr[n];
    //---
    return(max);
}
```

Данный шаблон описывает функцию, которая находит максимальное значение в переданном массиве и возвращает его в качестве результата. Напомним, что встроенная в MQL5 функция [ArrayMaximum\(\)](#) возвращает только индекс найденного максимального значения, по которому пользователь в дальнейшем может получить уже и само значение. Например, так:

```
//--- создадим массив
double array[];
int size=50;
ArrayResize(array,size);
//--- заполним случайными значениями
for(int i=0;i<size;i++)
    array[i]=MathRand();

//--- найдем позицию максимального элемента в массиве
int max_position=ArrayMaximum(array);
//--- теперь получим само максимальное значение в массиве
double max=array[max_position];
//--- вывод найденного значения
Print("Max value = ",max);
```

Таким образом, нам понадобилось два действия, чтобы получить максимальное значение в массиве. С помощью шаблона функции `ArrayMax()` можно получить результат нужного типа, просто передав в неё массив соответствующего типа. То есть вместо двух последних строчек

```
//--- найдем позицию максимального элемента в массиве
int max_position=ArrayMaximum(array);
//--- теперь получим само максимальное значение в массиве
double max=array[max_position];
```

теперь мы можем использовать одну строку, возвращающую сразу результат того же типа, что и переданный массив:

```
//--- найдем максимальное значение
double max=ArrayMax(array);
```

При этом тип результата, возвращенный функцией `ArrayMax()`, будет автоматически соответствовать типу массива.

Для создания универсальных способов работы с различными типами данных необходимо использовать ключевое слово `typename` для получения типа аргумента в виде строки. Покажем это на примере функции, которая возвращает в виде строки тип данных:

```
#include <Trade\Trade.mqh>
//+-----+
//| |
//+-----+
void OnStart()
{
//---
CTrade trade;
double d_value=M_PI;
int i_value=INT_MAX;
Print("d_value: type=",GetTypeName(d_value), ", value=", d_value);
Print("i_value: type=",GetTypeName(i_value), ", value=", i_value);
Print("trade: type=",GetTypeName(trade));
//---
}
//+-----+
//| Возвращает в строковом виде тип
//+-----+
template<typename T>
string GetTypeName(const T &t)
{
//--- вернем тип в виде строки
return(typename(T));
//---
}
```

Шаблоны функций можно также использовать и для методов класса, например:

```
class CFile
{
...
public:
...
template<typename T>
uint WriteStruct(T &data);
};

template<typename T>
uint CFile::WriteStruct(T &data)
{
```

```

...
return (FileWriteStruct(m_handle,data));
}

```

Шаблоны функций нельзя объявлять с ключевыми словами [export](#), [virtual](#) и [#import](#).

## Перегрузка шаблонных функций

В некоторых случаях может понадобиться перегрузка шаблонной функции. Например, у нас есть шаблонная функция, которая записывает в первый параметр значение второго параметра с помощью [явного приведения типов](#). В языке MQL5 запрещено приведение типа [string](#) к типу [bool](#), мы можем сделать это сами - для этого создадим перегрузку шаблонной функции. Например:

```

//+-----+
//| Шаблонная функция
//+-----+
template<typename T1,typename T2>
string Assign(T1 &var1,T2 var2)
{
    var1=(T1)var2;
    return (__FUNCSIG__);
}

//+-----+
//| Специальная перегрузка для случая bool+string
//+-----+
string Assign(bool &var1,string var2)
{
    var1=(StringCompare(var2,"true",false) || StringToInteger(var2)!=0);
    return (__FUNCSIG__);
}

//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    int i;
    bool b;
    Print(Assign(i,"test"));
    Print(Assign(b,"test"));
}

```

В результате выполнения данного кода мы увидим, что для пары `int+string` была использована шаблонная функция `Assign()`, а при втором вызове для пары `bool+string` уже использовалась перегруженная версия.

```

string Assign<int,string>(int&,string)
string Assign(bool&,string)

```

Смотри также

[Перегрузка](#)

## Чем хороши шаблоны

Шаблоны функций используются в тех случаях, когда необходимо производить одинаковые операции с данными разного типа, например - поиск максимального элемента в массиве. Главное преимущество использования шаблонов заключается в том, что программисту нет необходимости писать отдельную перегрузку для каждого типа. То есть вместо объявления множества перегрузок для каждого типа

```
double ArrayMax(double array[])
{
    ...
}

int ArrayMax(int array[])
{
    ...
}

uint ArrayMax(uint array[])
{
    ...
}

long ArrayMax(long array[])
{
    ...
}

datetime ArrayMax(datetime array[])
{
    ...
}
```

достаточно написать одну шаблонную функцию

```
template<typename T>
T ArrayMax(T array[])
{
    if(ArraySize()==0)
        return(0);
    uint max_index=ArrayMaximum(array);
    return(array[max_index]);
}
```

и затем использовать её в своем коде:

```
double high[];
datetime time[];
...
double max_high=ArrayMax(high);
datetime lasttime=ArrayMax(time);
```

Здесь формальный параметр *T*, который задает тип используемых данных, при компиляции заменяется на реально используемый тип, то есть компилятор автоматически генерирует

отдельную функцию под каждый тип - `double`, `datetime` и так далее. Точно также в языке MQL5 можно создавать шаблоны классов, используя все преимущества такого подхода.

## Шаблоны классов

Шаблон класса объявляется с помощью ключевого слова `template`, за которым идут угловые скобки `<>`, в которых перечисляется список формальных параметров с ключевым словом `typename`. Такая запись указывает компилятору, что перед ним обобщенный класс с формальным параметром `T`, задающим реальный тип переменной при реализации класса. Например, создадим класс-вектор для хранения массива с элементами типа `T`:

```
#define TOSTR(x) #x+" " // макрос для вывода имени объекта
//+-----+
//| Класс-вектор для хранения элементов типа Т |+
//+-----+
template <typename T>
class TArray
{
protected:
    T           m_array[];
public:
    //--- конструктор по умолчанию создает массив на 10 элементов
    void TArray(void){ArrayResize(m_array,10);}
    //--- конструктор для создания вектора с заданным размером массива
    void TArray(int size){ArrayResize(m_array,size);}
    //--- возвращает тип и количество данных, которые хранятся в объекте типа TArray
    string Type(void){return typename(m_array[0])+": "+(string)ArraySize(m_array);}
};
```

Далее, в программе создадим разными способами три объекта `TArray` для работы с разными типами

```
void OnStart()
{
    TArray<double> double_array; // вектор имеет размер по умолчанию 10
    TArray<int> int_array(15); // вектор имеет размер 15
    TArray<string> *string_array; // указатель на вектор TArray<string>
    //--- создадим динамический объект
    string_array=new TArray<string>(20);
    //--- выведем в Журнал имя объекта, тип данных и размер вектора
    PrintFormat("%s (%s)",TOSTR(double_array),double_array.Type());
    PrintFormat("%s (%s)",TOSTR(int_array),int_array.Type());
    PrintFormat("%s (%s)",TOSTR(string_array),string_array.Type());
    //--- удалим динамический объект перед завершением программы
    delete(string_array);
}
```

Результат выполнения скрипта:

```
double_array  (double:10)
int_array    (int:15)
```

```
string_array (string:20)
```

В результате было создано 3 вектора с разными типами данных: double, int и string.

Шаблоны классов хорошо подходят для разработки контейнеров - объектов, которые предназначены для инкапсуляции объектов любого типа. Объектами контейнеров являются коллекции, которые уже содержат объекты одного определенного типа. Как правило в контейнер сразу же встраивается и реализация по работе с данными, которые в нём хранятся.

Например, можно создать шаблон класса, который не позволяет обратиться к элементу за пределами массива, и таким образом избегать [критической ошибки "out of range"](#).

```
//+-----+
//| Класс для безопасного обращения к элементу массива |
//+-----+
template<typename T>
class TSafeArray
{
protected:
    T             m_array[];
public:
    //--- конструктор по умолчанию
    void          TSafeArray(void) {}
    //--- конструктор для создания массива заданного размера
    void          TSafeArray(int size){ArrayResize(m_array,size);}
    //--- размер массива
    int           Size(void){return(ArraySize(m_array));}
    //--- изменение размера массива
    int           Resize(int size,int reserve){return(ArrayResize(m_array,size,reserve));}
    //--- освобождение массива
    void          Erase(void){ZeroMemory(m_array);}
    //--- оператор доступа к элементу массива по индексу
    T             operator[](int index);
    //--- оператор присваивания для получения сразу всех элементов из массива
    void          operator=(const T &array[]); // массив типа T
};

//+-----+
//| Операция получения элемента по индексу |
//+-----+
template<typename T>
T TSafeArray::operator[](int index)
{
    static T invalid_value;
    //---
    int max=ArraySize(m_array)-1;
    if(index<0 || index>=ArraySize(m_array))
    {
        PrintFormat("%s index %d is not in range (0-%d)!",__FUNCTION__,index,max);
        return(invalid_value);
    }
}
```

```

//---
    return(m_array[index]);
}
//+-----+
//| Операция присваивания для массива |
//+-----+
template<typename T>
void TSafeArray::operator=(const T &array[])
{
    int size=ArraySize(array);
    ArrayResize(m_array,size);
//--- тип Т должен поддерживать оператор копирования
    for(int i=0;i<size;i++)
        m_array[i]=array[i];
//---
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    int copied,size=15;
    MqlRates rates[];
//--- скопируем массив котировок
    if((copied=CopyRates(_Symbol,_Period,0,size,rates))!=size)
    {
        PrintFormat("CopyRates(%s,%s,0,%d) вернула код ошибки %d",
                   _Symbol,EnumToString(_Period),size,GetLastError());
        return;
    }
//--- создадим контейнер и вложим в него массив значений MqlRates
    TSafeArray<MqlRates> safe_rates;
    safe_rates=rates;
//--- индекс в пределах массива
    int index=3;
    PrintFormat("Close[%d]=%G",index,safe_rates[index].close);
//--- индекс за пределами массива
    index=size;
    PrintFormat("Close[%d]=%G",index,safe_rates[index].close);
}

```

Обратите внимание, что при описании методов за пределами декларации класса также необходимо использовать объявление шаблона:

```

template<typename T>
T TSafeArray::operator[](int index)
{
    ...
}

```

```
template<typename T>
void TSafeArray::operator=(const T &array[])
{
    ...
}
```

Шаблоны классов и функций позволяют указывать несколько формальных параметров через запятую, например, коллекция Map для хранения пар "ключ - значение":

```
template<typename Key, template Value>
class TMap
{
    ...
}
```

#### Смотри также

[Шаблоны функций](#), [Перегрузка](#)

## Абстрактные классы и чисто виртуальные функции

Абстрактные классы предназначены для создания обобщенных сущностей, на основе которых в дальнейшем предполагается создавать более конкретные производные классы. Абстрактный класс - это класс, который может использоваться лишь в качестве базового класса для некоторого другого класса, поэтому невозможно создать объект типа абстрактного класса.

Класс, содержащий хотя бы одну чисто виртуальную функцию, является абстрактным. Поэтому, классы, производные от абстрактного класса, должны реализовать все его чисто виртуальные функции, иначе они также будут абстрактными классами.

Виртуальная функция объявляется как "чистая" с помощью синтаксиса спецификатора-чистоты. Рассмотрим в качестве примера класс CAnimal, который создаётся только для того, чтобы предоставлять общие функции - сами объекты типа CAnimal имеют слишком общий характер для практического применения. Таким образом, класс CAnimal является хорошим кандидатом в абстрактный класс:

```
class CAnimal
{
public:
    CAnimal();           // конструктор
    virtual void Sound() = 0; // чисто виртуальная функция
private:
    double m_legs_count; // количество ног животного
};
```

Здесь функция Sound() является чисто виртуальной, потому что она объявлена со спецификатором чисто виртуальной функции PURE (=0).

Чисто виртуальными функциями являются только такие виртуальные функции, для которых указан спецификатор чистоты PURE, а именно: (=NULL) или (=0). Пример объявления и использования абстрактного класса:

```
class CAnimal
{
public:
    virtual void Sound() =NULL; // PURE method, должен быть переопределён в потомке
};

//--- ПОТОМОК от абстрактного класса
class CCat : public CAnimal
{
public:
    virtual void Sound() { Print("Myau"); } // PURE переопределён, класс CCat не является абстрактным
};

//--- примеры неправильного использования
new CAnimal;           // ошибка 'CAnimal' - компилятор выдаст ошибку "cannot instantiate abstract class"
CAnimal some_animal; // ошибка 'CAnimal' - компилятор выдаст ошибку "cannot instantiate abstract class"

//--- примеры правильного использования
new CCat; // ошибки нет - класс CCat не абстрактный
```

```
CCat cat; // ошибки нет - класс CCat не абстрактный
```

## Ограничения на использование абстрактных классов

При вызове конструктором абстрактного класса чистой виртуальной функции (прямо или косвенно) результат будет неопределённым.

```
//+-----+
//| Абстрактный базовый класс |
//+-----+
class CAnimal
{
public:
    //--- чисто виртуальная функция
    virtual void      Sound(void)=NULL;
    //--- функция
    void             CallSound(void) { Sound(); }
    //--- конструктор
    CAnimal()
    {
        //--- явный вызов виртуального метода
        Sound();
        //--- неявный вызов (через третью функцию)
        CallSound();
        //--- в конструкторе и/или деструкторе всегда вызываются свои функции,
        //--- несмотря на виртуальность и переопределение вызываемой функции в потомке
        //--- если вызываемая функция чисто виртуальная, то
        //--- вызов приведёт к критической ошибке выполнения: "pure virtual function cal
    }
};
```

Однако конструкторы и деструкторы абстрактных классов могут вызывать другие функции-члены.

## Стандартные константы, перечисления и структуры

Для облегчения написания программ, а также для удобства восприятия исходных текстов программ, в языке MQL5 предусмотрены предопределенные стандартные константы и перечисления. Кроме того, для хранения информации используются служебные [структуры](#).

Стандартные константы являются аналогом макроподстановок и имеют тип [int](#).

Константы сгруппированы по своему назначению:

- [Константы графиков](#) используются при работе с ценовыми графиками: открытие, навигация, установка параметров;
- [Константы объектов](#) предназначены для обработки графических объектов, которые можно создавать и отображать на графиках;
- [Константы индикаторов](#) служат для работы со стандартными и пользовательскими индикаторами;
- [Состояние окружения](#) - описывают свойства mq5-программы, предоставляют информацию о клиентском терминале, торговом инструменте и текущем торговом счете;
- [Торговые константы](#) позволяют уточнять разнообразную информацию в процессе торговли;
- [Именованные константы](#) - константы языка MQL5;
- [Структуры данных](#) описывают используемые форматы хранения данных;
- [Коды ошибок и предупреждений](#) описывают сообщения компилятора и сообщения торгового сервера на торговые запросы;
- [Константы ввода/вывода](#) предназначены для работы с [файловыми функциями](#) и вывода сообщений на экран компьютера функцией [MessageBox\(\)](#).

## Константы графиков

Константы, описывающие различные свойства графиков, разделены на следующие группы:

- [Типы событий](#) - события, которые возникают при работе с графиком;
- [Периоды графиков](#) - стандартные встроенные периоды;
- [Свойства графиков](#) - идентификаторы, используемые в качестве параметров функций для работы с графиками;
- [Константы позиционирования](#) - значения параметра функции [ChartNavigate\(\)](#);
- [Отображение графиков](#) - задание внешнего вида графика.

## Типы событий графика

Существуют 11 видов событий, которые можно обрабатывать с помощью функции предопределенной функции [OnChartEvent\(\)](#). Для пользовательских событий предусмотрено 65535 идентификаторов в диапазоне от CHARTEVENT\_CUSTOM до CHARTEVENT\_CUSTOM\_LAST включительно. Для генерации пользовательского события необходимо использовать функцию [EventChartCustom\(\)](#).

### ENUM\_CHART\_EVENT

Идентификатор	Описание
CHARTEVENT_KEYDOWN	Нажатие клавиатуры
CHARTEVENT_MOUSE_MOVE	Перемещение мыши и нажатие кнопок мышки (если для графика установлено свойство <a href="#">CHART_EVENT_MOUSE_MOVE=true</a> )
CHARTEVENT_MOUSE_WHEEL	Нажатие или прокрутка колесика мышки (если для графика установлено свойство <a href="#">CHART_EVENT_MOUSE_WHEEL=true</a> )
CHARTEVENT_OBJECT_CREATE	Создание <a href="#">графического объекта</a> (если для графика установлено свойство <a href="#">CHART_EVENT_OBJECT_CREATE=true</a> )
CHARTEVENT_OBJECT_CHANGE	Изменение свойств <a href="#">графического объекта</a> через диалог свойств
CHARTEVENT_OBJECT_DELETE	Удаление <a href="#">графического объекта</a> (если для графика установлено свойство <a href="#">CHART_EVENT_OBJECT_DELETE=true</a> )
CHARTEVENT_CLICK	Нажатие мышки на графике
CHARTEVENT_OBJECT_CLICK	Нажатие мышки на <a href="#">графическом объекте</a>
CHARTEVENT_OBJECT_DRAG	Перетаскивание <a href="#">графического объекта</a>
CHARTEVENT_OBJECT_ENDEDIT	Окончание редактирования текста в <a href="#">графическом объекте Edit</a>
CHARTEVENT_CHART_CHANGE	Изменение размеров графика или изменение свойств графика через диалог свойств
CHARTEVENT_CUSTOM	Начальный номер события из диапазона пользовательских событий
CHARTEVENT_CUSTOM_LAST	Конечный номер события из диапазона пользовательских событий

Для каждого типа события входные параметры функции [OnChartEvent\(\)](#) имеют определенные значения, которые необходимы для обработки этого события. В таблице перечислены события и значения, которые передаются через параметры.

Событие	Значение параметра id	Значение параметра lparam	Значение параметра dparam	Значение параметра sparam
Событие нажатия клавиатуры	CHARTEVENT_KEYDOWN	код нажатой клавиши	Количество нажатий клавиши, сгенерированных за время её удержания в нажатом состоянии	Строковое значение битовой маски, описывающее статус кнопок клавиатуры
События мышки (если для графика установлено свойство <code>CHART_EVENT_MOUSE_MOVE=true</code> )	CHARTEVENT_MOUSE_MOVE	X координата	Y координата	Строковое значение битовой маски, описывающее статус кнопок мыши
Событие колёсика мышки (если для графика установлено свойство <code>CHART_EVENT_MOUSE_WHEEL=true</code> )	CHARTEVENT_MOUSE_WHEEL	Флаги состояний клавиш и кнопок мышки, координаты X и Y курсора. Описание дано в <a href="#">примере ниже</a>	Значение прокрутки колёсика мышки	—
Событие создания графического объекта (если для графика установлено свойство <code>CHART_EVENT_OBJECT_CREATE=true</code> )	CHARTEVENT_OBJECT_CREATE	—	—	Имя созданного графического объекта
Событие изменения свойств объекта через диалог свойств	CHARTEVENT_OBJECT_CHANGE	—	—	Имя измененного графического объекта
Событие удаления графического объекта (если	CHARTEVENT_OBJECT_DELETE	—	—	Имя удаленного графического объекта

для графика установлено свойство <a href="#">CHART_EVENT_OBJECT_DELETE=true</a> )				
Событие щелчка мышки на графике	CHARTEVENT_CLICK	X координата	Y координата	—
Событие щелчка мышки на графическом объекте	CHARTEVENT_OBJECT_CLICK	X координата	Y координата	Имя графического объекта, на котором произошло событие
Событие перемещения графического объекта при помощи мышки	CHARTEVENT_OBJECT_DRAG	—	—	Имя перемещенного графического объекта
Событие окончания редактирования текста в поле ввода графического объекта "Поле ввода"	CHARTEVENT_OBJECT_ENDEDIT	—	—	Имя графического объекта "Поле ввода", в котором завершилось редактирование текста
Событие изменения размеров графика или изменения свойств графика через диалог свойств	CHARTEVENT_CHART_CHANGE	—	—	—
Пользовательское событие с номером N	CHARTEVENT_CUSTOM+N	Значение, заданное функцией <a href="#">EventChartCustom()</a>	Значение, заданное функцией <a href="#">EventChartCustom()</a>	Значение, заданное функцией <a href="#">EventChartCustom()</a>

Пример:

```
#define KEY_NUMPAD_5      12
#define KEY_LEFT           37
#define KEY_UP             38
```

```

#define KEY_RIGHT          39
#define KEY_DOWN           40
#define KEY_NUMLOCK_DOWN   98
#define KEY_NUMLOCK_LEFT   100
#define KEY_NUMLOCK_5       101
#define KEY_NUMLOCK_RIGHT  102
#define KEY_NUMLOCK_UP      104

//+-----+
//| Expert initialization function               |
//+-----+
int OnInit()
{
//---
    Print("Запущен эксперт с именем ",MQL5InfoString(MQL5_PROGRAM_NAME));
//--- установка флага получения событий создания объектов графика
    ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_CREATE,true);
//--- установка флага получения событий удаления объектов графика
    ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_DELETE,true);
//--- принудительное обновление свойств графика гарантирует готовность к обработке соо
    ChartRedraw();
//---
    return(INIT_SUCCEEDED);
}

//+-----+
//| ChartEvent function                         |
//+-----+
void OnChartEvent(const int id,           // идентификатор события
                  const long& lparam,     // параметр события типа long
                  const double& dparam,   // параметр события типа double
                  const string& sparam)  // параметр события типа string
)
{
//--- нажатие левой кнопкой мышки на графике
    if(id==CHARTEVENT_CLICK)
        Print("Координаты щелчка мышки на графике: x = ",lparam," y = ",dparam);
//--- нажатие мышкой на графическом объекте
    if(id==CHARTEVENT_OBJECT_CLICK)
        Print("Нажатие кнопки мышки на объекте с именем '"+sparam+"'");
//--- нажатие кнопки на клавиатуре
    if(id==CHARTEVENT_KEYDOWN)
    {
        switch(lparam)
        {
            case KEY_NUMLOCK_LEFT: Print("Нажата KEY_NUMLOCK_LEFT"); break;
            case KEY_LEFT:         Print("Нажата KEY_LEFT"); break;
            case KEY_NUMLOCK_UP:   Print("Нажата KEY_NUMLOCK_UP"); break;
            case KEY_UP:           Print("Нажата KEY_UP"); break;
            case KEY_NUMLOCK_RIGHT:Print("Нажата KEY_NUMLOCK_RIGHT"); break;
            case KEY_RIGHT:        Print("Нажата KEY_RIGHT"); break;
        }
    }
}

```

```

        case KEY_NUMLOCK_DOWN: Print("Нажата KEY_NUMLOCK_DOWN"); break;
        case KEY_DOWN: Print("Нажата KEY_DOWN"); break;
        case KEY_NUMPAD_5: Print("Нажата KEY_NUMPAD_5"); break;
        case KEY_NUMLOCK_5: Print("Нажата KEY_NUMLOCK_5"); break;
        default: Print("Нажата какая-то неперечисленная клавиша");
    }
    ChartRedraw();
}
//--- удален объекта
if(id==CHARTEVENT_OBJECT_DELETE)
    Print("Удален объект с именем ",sparam);
//--- создан объект
if(id==CHARTEVENT_OBJECT_CREATE)
    Print("Создан объект с именем ",sparam);
//--- перемещен объект или изменены координаты точек привязки
if(id==CHARTEVENT_OBJECT_DRAG)
    Print("Изменение точек привязки объекта с именем ",sparam);
//--- изменен текст в поле ввода графического объекта Edit
if(id==CHARTEVENT_OBJECT_ENDEDIT)
    Print("Изменен текст в объекте Edit ",sparam);
}

```

Для события CHARTEVENT\_MOUSE\_MOVE строковой параметр **sparam** содержит число, представляющее информацию о состоянии клавиш:

Бит	Описание
1	Состояние левой клавиши мыши
2	Состояние правой клавиши мыши
3	Состояние клавиши SHIFT
4	Состояние клавиши CTRL
5	Состояние средней клавиши мыши
6	Состояние первой дополнительной клавиши мыши
7	Состояние второй дополнительной клавиши мыши

### Пример:

```

//-----+
//| Expert initialization function
//-----+
void OnInit()
{
//--- включение сообщений о перемещении мыши по окну чарта
    ChartSetInteger(0,CHART_EVENT_MOUSE_MOVE,1);
//--- принудительное обновление свойств графика гарантирует готовность к обработке соо

```

```

    ChartRedraw();
}

//+-----+
//| MouseState
//+-----+
string MouseState(uint state)
{
    string res;
    res+="\nML: " + (((state& 1)== 1)? "DN":"UP"); // mouse left
    res+="\nMR: " + (((state& 2)== 2)? "DN":"UP"); // mouse right
    res+="\nMM: " + (((state&16)==16)? "DN":"UP"); // mouse middle
    res+="\nMX: " + (((state&32)==32)? "DN":"UP"); // mouse first X key
    res+="\nMY: " + (((state&64)==64)? "DN":"UP"); // mouse second X key
    res+="\nSHIFT: "+(((state& 4)== 4)? "DN":"UP"); // shift key
    res+="\nCTRL: " + (((state& 8)== 8)? "DN":"UP"); // control key
    return(res);
}

//+-----+
//| ChartEvent function
//+-----+
void OnChartEvent(const int id,const long &lparam,const double &dparam,const string &s
{
    if(id==CHARTEVENT_MOUSE_MOVE)
        Comment("POINT: ",(int)lparam,",",(int)dparam,"\\n",MouseState((uint)sparam));
}

```

Для события CHARTEVENT\_MOUSE\_WHEEL параметры **lparam** и **dparam** содержат информацию о состоянии клавиш **Ctrl**, **Shift**, кнопок мышки, координатах курсора и величине прокрутки колёсика мышки. Для лучшего понимания запустите этот советник на графике и прокручивайте колёсико мышки, нажимая поочередно различные кнопки и клавиши, описанные в коде.

**Пример** обработки события CHARTEVENT\_MOUSE\_WHEEL:

```

//+-----+
//| Expert initialization function
//+-----+
init OnInit()
{
    //--- включение сообщений о прокрутке колесика мышки
    ChartSetInteger(0,CHART_EVENT_MOUSE_WHEEL,1);
    //--- принудительное обновление свойств графика гарантирует готовность к обработке событий
    ChartRedraw();
    //---
    return(INIT_SUCCEEDED);
}

//+-----+
//| ChartEvent function
//+-----+

```

```

void OnChartEvent(const int id,const long &lparam,const double &dparam,const string &s)
{
    if(id==CHARTEVENT_MOUSE_WHEEL)
    {
        //--- разберем состояние кнопок и колесика мышки для этого события
        int flg_keys = (int)(lparam>>32);           // флаг состояний клавиш Ctrl, Shift
        int x_cursor = (int)(short)lparam;             // X-координата, в которой произошло
        int y_cursor = (int)(short)(lparam>>16);      // Y-координата, в которой произошло
        int delta     = (int)dparam;                   // суммарное значение прокрутки колеса

        //--- обрабатаем флаг
        string str_keys="";
        if((flg_keys&0x0001)!=0)
            str_keys+="LMOUSE ";
        if((flg_keys&0x0002)!=0)
            str_keys+="RMOUSE ";
        if((flg_keys&0x0004)!=0)
            str_keys+="SHIFT ";
        if((flg_keys&0x0008)!=0)
            str_keys+="CTRL ";
        if((flg_keys&0x0010)!=0)
            str_keys+="MMOUSE ";
        if((flg_keys&0x0020)!=0)
            str_keys+="X1MOUSE ";
        if((flg_keys&0x0040)!=0)
            str_keys+="X2MOUSE ";

        if(str_keys!="")
            str_keys=", keys='"+StringSubstr(str_keys,0,StringLen(str_keys)-1)+'";
        PrintFormat ("%s: X=%d, Y=%d, delta=%d%s",EnumToString(CHARTEVENT_MOUSE_WHEEL),x_
        }
    }
//+-----+ /*

```

**Пример вывода**

```

CHARTEVENT_MOUSE_WHEEL: Ctrl pressed: X=193, Y=445, delta=-120
CHARTEVENT_MOUSE_WHEEL: Shift pressed: X=186, Y=446, delta=120
CHARTEVENT_MOUSE_WHEEL:   X=178, Y=447, delta=-120
CHARTEVENT_MOUSE_WHEEL:   X=231, Y=449, delta=120
CHARTEVENT_MOUSE_WHEEL: MiddleButton pressed: X=231, Y=449, delta=120
CHARTEVENT_MOUSE_WHEEL: LeftButton pressed: X=279, Y=320, delta=-120
CHARTEVENT_MOUSE_WHEEL: RightButton pressed: X=253, Y=330, delta=120 */

```

## Смотри также

[Функции обработки событий](#), [Работа с событиями](#)

## Периоды графиков

Все предопределенные периоды графиков имеют уникальные идентификаторы. Идентификатор PERIOD\_CURRENT означает текущий период графика, на котором запущена mq5-программа.

### ENUM\_TIMEFRAMES

Идентификатор	Описание
PERIOD_CURRENT	Текущий период
PERIOD_M1	1 минута
PERIOD_M2	2 минуты
PERIOD_M3	3 минуты
PERIOD_M4	4 минуты
PERIOD_M5	5 минут
PERIOD_M6	6 минут
PERIOD_M10	10 минут
PERIOD_M12	12 минут
PERIOD_M15	15 минут
PERIOD_M20	20 минут
PERIOD_M30	30 минут
PERIOD_H1	1 час
PERIOD_H2	2 часа
PERIOD_H3	3 часа
PERIOD_H4	4 часа
PERIOD_H6	6 часов
PERIOD_H8	8 часов
PERIOD_H12	12 часов
PERIOD_D1	1 день
PERIOD_W1	1 неделя
PERIOD_MN1	1 месяц

**Пример:**

```
string chart_name="test_Object_Chart";
Print("Попробуем создать объект Chart с именем ",chart_name);
//--- если такого объекта нет - создадим его
if(ObjectFind(0,chart_name)<0)ObjectCreate(0,chart_name,OBJ_CHART,0,0,0,0,0);
```

```

//--- зададим символ
ObjectSetString(0,chart_name,OBJPROP_SYMBOL,"EURUSD");
//--- зададим координату X для точки привязки
ObjectSetInteger(0,chart_name,OBJPROP_XDISTANCE,100);
//--- зададим координату Y для точки привязки
ObjectSetInteger(0,chart_name,OBJPROP_YDISTANCE,100);
//--- установим ширину
ObjectSetInteger(0,chart_name,OBJPROP_XSIZE,400);
//--- установим высоту
ObjectSetInteger(0,chart_name,OBJPROP_YSIZE,300);
//--- установим период
ObjectSetInteger(0,chart_name,OBJPROP_PERIOD,PERIOD_D1);
//--- установим масшаб ( от 0 до 5)
ObjectSetDouble(0,chart_name,OBJPROP_SCALE,4);
//--- сделаем недоступным для выделения мышкой
ObjectSetInteger(0,chart_name,OBJPROP_SELECTABLE,false);

```

## Идентификаторы таймсерий

Идентификаторы таймсерий используются в функциях [iHighest\(\)](#) и [iLowest\(\)](#). Могут быть одним из значений перечисления

### ENUM\_SERIESMODE

Идентификатор	Описание
MODE_OPEN	Цена открытия
MODE_LOW	Минимальная цена
MODE_HIGH	Максимальная цена
MODE_CLOSE	Цена закрытия
MODE_VOLUME	Тиковый объем
MODE_REAL_VOLUME	Реальный объем
MODE_SPREAD	Спред

### Смотри также

[PeriodSeconds](#), [Period](#), [Дата и время](#), [Видимость объектов](#)

## Свойства графиков

Идентификаторы семейства перечислений ENUM\_CHART\_PROPERTY используются в качестве параметров [функций для работы с графиками](#). Аббревиатура r/o в столбце "Тип свойства" означает, что данное свойство предназначено только для чтения и не может быть изменено. Аббревиатура w/o в столбце "Тип свойства" означает, что данное свойство предназначено только для записи и не может быть получено. При обращении к некоторым свойствам необходимо указывать дополнительный параметр-модификатор (modifier), который служит для указания номера подокна графика. 0 означает главное окно.

Функции, устанавливающие свойства графика, фактически служат для отправки ему команд на изменение. При успешном выполнении этих функций команда попадает в общую очередь событий графика. Изменение графика производится в процессе обработки очереди событий данного графика.

По этой причине не следует ожидать немедленного визуального обновления графика после вызова данных функций. В общем случае обновление графика производится терминалом автоматически по событиям изменения - поступление новой котировки, изменения размера окна графика и т.д. Для принудительного обновления внешнего вида графика используйте команду на перерисовку графика [ChartRedraw\(\)](#).

Для функций [ChartSetInteger\(\)](#) и [ChartGetInteger\(\)](#)

### ENUM\_CHART\_PROPERTY\_INTEGER

Идентификатор	Описание	Тип свойства
CHART_SHOW	<p>Признак отрисовки ценового графика. Если установлено значение <code>false</code>, то отключается отрисовка любых атрибутов ценового графика и устраняются все отступы по краям графика: шкалы времени и цены, строка быстрой навигации, метки событий Календаря, значки сделок, тултипы индикаторов и баров, подокна индикаторов, гистограммы объёмов и т.д. Отключение отрисовки является идеальным решением для создания собственного интерфейса программы с использованием графических <a href="#">ресурсов</a>. <a href="#">Графические объекты</a> отрисовываются всегда независимо от</p>	bool

	установленного значения свойства CHART_SHOW.	
<a href="#"><u>CHART_IS_OBJECT</u></a>	Признак идентификации объекта "График" ( <a href="#">OBJ_CHART</a> ) - для графического объекта возвращает true. Для настоящего графика значение равно false	bool r/o
<a href="#"><u>CHART_BRING_TO_TOP</u></a>	Показ графика поверх всех других	bool w/o
CHART_CONTEXT_MENU	Включение/отключение доступа к контекстному меню по нажатию правой клавиши мышки. Значение CHART_CONTEXT_MENU=false отключает только контекстное меню графика, при этом контекстное меню для объектов на графике остается доступным.	bool (значение по умолчанию true)
CHART_CROSSHAIR_TOOL	Включение/отключение доступа к инструменту "Перекрестье" по нажатию средней клавиши мышки	bool (значение по умолчанию true)
<a href="#"><u>CHART_MOUSE_SCROLL</u></a>	Прокрутка графика левой кнопкой мышки по горизонтали. Прокрутка по вертикали также будет доступна, если установлено в true значение любого из трех свойств: CHART_SCALEFIX, CHART_SCALEFIX_11 или CHART_SCALE_PT_PER_BAR При CHART_MOUSE_SCROLL=false прокрутка графика колёсиком мышки будет недоступна	bool
CHART_EVENT_MOUSE_WHEEL	Отправка всем mql5-программам на графике сообщений о событиях колёсика мышки ( <a href="#">CHARTEVENT_MOUSE_WHEEL</a> )	bool (значение по умолчанию true)

<a href="#"><u>CHART_EVENT_MOUSE_MOVE</u></a>	Отправка всем mql5-программам на графике сообщений о событиях перемещения и нажатия кнопок мышки ( <a href="#">CHARTEVENT_MOUSE_MOVE</a> )	bool
<a href="#"><u>CHART_EVENT_OBJECT_CREATE</u></a>	Отправка всем mql5-программам на графике сообщений о событии создания графического объекта ( <a href="#">CHARTEVENT_OBJECT_CREATE</a> )	bool
<a href="#"><u>CHART_EVENT_OBJECT_DELETE</u></a>	Отправка всем mql5-программам на графике сообщений о событии удаления графического объекта ( <a href="#">CHARTEVENT_OBJECT_DELETE</a> )	bool
<a href="#"><u>CHART_MODE</u></a>	Тип графика (свечи, бары или линия)	enum <a href="#"><u>ENUM_CHART_MODE</u></a>
<a href="#"><u>CHART_FOREGROUND</u></a>	Ценовой график на переднем плане	bool
<a href="#"><u>CHART_SHIFT</u></a>	Режим отступа ценового графика от правого края	bool
<a href="#"><u>CHART_AUTOSCROLL</u></a>	Режим автоматического перехода к правому краю графика	bool
<a href="#"><u>CHART_KEYBOARD_CONTROL</u></a>	Разрешение на управление графиком с помощью клавиатуры ("Home", "End", "PageUp", "+", "-", "Стрелка вверх" и т.д.). Установка CHART_KEYBOARD_CONTROL=false позволяет отключить скроллинг и масштабирование графика, но при этом сохраняется возможность получать события нажатия данных клавиш в <a href="#">OnChartEvent()</a> .	bool
<a href="#"><u>CHART_QUICK_NAVIGATION</u></a>	Разрешение на перехват графиком нажатий клавиш Space и Enter для активации	bool

	строки быстрой навигации. Стока быстрой навигации автоматически появляется внизу графика при двойном клике мышки или нажатии клавиш Space/Enter. Таким образом можно быстро сменить символ, таймфрейм и дату первого видимого бара.	
<a href="#"><u>CHART_SCALE</u></a>	Масштаб	int от 0 до 5
<a href="#"><u>CHART_SCALEFIX</u></a>	Режим фиксированного масштаба	bool
<a href="#"><u>CHART_SCALEFIX_11</u></a>	Режим масштаба 1:1	bool
<a href="#"><u>CHART_SCALE_PT_PER_BAR</u></a>	Режим указания масштаба в пунктах на бар	bool
<a href="#"><u>CHART_SHOW_OHLC</u></a>	Отображение в левом верхнем углу значений OHLC	bool
<a href="#"><u>CHART_SHOW_BID_LINE</u></a>	Отображение значения Bid горизонтальной линией на графике	bool
<a href="#"><u>CHART_SHOW_ASK_LINE</u></a>	Отображение значения Ask горизонтальной линией на графике	bool
<a href="#"><u>CHART_SHOW_LAST_LINE</u></a>	Отображение значения Last горизонтальной линией на графике	bool
<a href="#"><u>CHART_SHOW_PERIOD_SEP</u></a>	Отображение вертикальных разделителей между соседними периодами	bool
<a href="#"><u>CHART_SHOW_GRID</u></a>	Отображение сетки на графике	bool
<a href="#"><u>CHART_SHOW_VOLUMES</u></a>	Отображение объемов на графике	enum <a href="#"><u>ENUM_CHART_VOLUME_MODE</u></a>
<a href="#"><u>CHART_SHOW_OBJECT_DESCR</u></a>	Отображение текстовых описаний объектов (не для всех объектов описания показываются)	bool
<a href="#"><u>CHART_VISIBLE_BARS</u></a>	Количество баров на графике, доступных для отображения	int r/o

<a href="#"><u>CHART_WINDOWS_TOTAL</u></a>	Общее количество окон графика, включая подокна индикаторов	int r/o
<a href="#"><u>CHART_WINDOW_IS_VISIBLE</u></a>	Видимость подокон	bool r/o модификатор - номер подокна
<a href="#"><u>CHART_WINDOW_HANDLE</u></a>	Хэндл графика (HWND)	int r/o
<a href="#"><u>CHART_WINDOW_YDISTANCE</u></a>	<p>Дистанция в пикселях по вертикальной оси Y между верхней рамкой подокна индикатора и верхней рамкой главного окна графика. При наступлении событий мыши координаты курсора передаются в координатах главного окна графика, в то время как координаты графических объектов в подокне индикатора задаются относительно верхнего левого угла подокна.</p> <p>Значение требуется для перевода абсолютных координат главного графика в локальные координаты подокна для корректной работы с графическими объектами, у которых координаты задаются относительно верхнего левого угла рамки подокна.</p>	int r/o модификатор - номер подокна
<a href="#"><u>CHART_FIRST_VISIBLE_BAR</u></a>	Номер первого видимого бара на графике. Индексация баров соответствует <a href="#">таймсерии</a> .	int r/o
<a href="#"><u>CHART_WIDTH_IN_BARS</u></a>	Ширина графика в барах	int r/o
<a href="#"><u>CHART_WIDTH_IN_PIXELS</u></a>	Ширина графика в пикселях	int r/o
<a href="#"><u>CHART_HEIGHT_IN_PIXELS</u></a>	Высота графика в пикселях	int модификатор - номер подокна
<a href="#"><u>CHART_COLOR_BACKGROUND</u></a>	Цвет фона графика	color
<a href="#"><u>CHART_COLOR_FOREGROUND</u></a>	Цвет осей, шкалы и строки OHLC	color
<a href="#"><u>CHART_COLOR_GRID</u></a>	Цвет сетки	color

<u>CHART_COLOR_VOLUME</u>	Цвет объемов и уровней открытия позиций	color
<u>CHART_COLOR_CHART_UP</u>	Цвет бара вверх, тени и окантовки тела бычьей свечи	color
<u>CHART_COLOR_CHART_DOWN</u>	Цвет бара вниз, тени и окантовки тела медвежьей свечи	color
<u>CHART_COLOR_CHART_LINE</u>	Цвет линии графика и японских свечей "Доджи"	color
<u>CHART_COLOR_CANDLE_BULL</u>	Цвет тела бычьей свечи	color
<u>CHART_COLOR_CANDLE_BEAR</u>	Цвет тела медвежьей свечи	color
<u>CHART_COLOR_BID</u>	Цвет линии Bid-цены	color
<u>CHART_COLOR_ASK</u>	Цвет линии Ask-цены	color
<u>CHART_COLOR_LAST</u>	Цвет линии цены последней совершенной сделки (Last)	color
<u>CHART_COLOR_STOP_LEVEL</u>	Цвет уровней стоп-ордеров (Stop Loss и Take Profit)	color
<u>CHART_SHOW_TRADE_LEVELS</u>	Отображение на графике торговых уровней (уровни открытых позиций, Stop Loss, Take Profit и отложенных ордеров)	bool
<u>CHART_DRAG_TRADE_LEVELS</u>	Разрешение на перетаскивание торговых уровней на графике с помощью мышки. По умолчанию режим перетаскивания включен (значение true)	bool
<u>CHART_SHOW_DATE_SCALE</u>	Отображение на графике шкалы времени	bool
<u>CHART_SHOW_PRICE_SCALE</u>	Отображение на графике ценовой шкалы	bool
<u>CHART_SHOW_ONE_CLICK</u>	Отображение на графике панели быстрой торговли (опция " <a href="#">Торговля одним кликом</a> ")	bool
<u>CHART_IS_MAXIMIZED</u>	Окно графика развернуто	bool
<u>CHART_IS_MINIMIZED</u>	Окно графика свернуто	bool

CHART_IS_DOCKED	Окно графика закреплено. Если установить значение <b>false</b> , то график можно перетащить за пределы терминала	bool
CHART_FLOAT_LEFT	Левая координата открепленного графика относительно виртуального экрана	int
CHART_FLOAT_TOP	Верхняя координата открепленного графика относительно виртуального экрана	int
CHART_FLOAT_RIGHT	Правая координата открепленного графика относительно виртуального экрана	int
CHART_FLOAT_BOTTOM	Нижняя координата открепленного графика относительно виртуального экрана	int

Для функций [ChartSetDouble\(\)](#) и [ChartGetDouble\(\)](#)

#### ENUM\_CHART\_PROPERTY\_DOUBLE

Идентификатор	Описание	Тип свойства
<a href="#">CHART_SHIFT_SIZE</a>	Размер отступа нулевого бара от правого края в процентах	double (от 10 до 50 процентов)
<a href="#">CHART_FIXED_POSITION</a>	Положение фиксированной позиции графика от левого края в процентах. Фиксированная позиция графика обозначена маленьким серым треугольником на горизонтальной оси времени и показывается только в том случае, если отключена автоматическая прокрутка к правому краю при поступлении нового тика (смотри свойство CHART_AUTOSCROLL). Бар, который находится на фиксированной позиции, остаётся на том же месте при	double

	увеличении и уменьшении масштаба.	
<u>CHART_FIXED_MAX</u>	Фиксированный максимум графика	double
<u>CHART_FIXED_MIN</u>	Фиксированный минимум графика	double
<u>CHART_POINTS_PER_BAR</u>	Значение масштаба в пунктах на бар	double
<u>CHART_PRICE_MIN</u>	Минимум графика	double r/o модификатор - номер подокна
<u>CHART_PRICE_MAX</u>	Максимум графика	double r/o модификатор - номер подокна

Для функций [ChartSetString\(\)](#) и [ChartGetString\(\)](#)

#### ENUM\_CHART\_PROPERTY\_STRING

Идентификатор	Описание	Тип свойства
<u>CHART_COMMENT</u>	Текст комментария на графике	string
<u>CHART_EXPERT_NAME</u>	Имя эксперта, запущенного на графике с указанным chart_id	string
<u>CHART_SCRIPT_NAME</u>	Имя скрипта, запущенного на графике с указанным chart_id	string

**Пример:**

```

int chartMode=ChartGetInteger(0,CHART_MODE);
switch(chartMode)
{
    case(CHART_BARS):    Print("CHART_BARS");    break;
    case(CHART_CANDLES): Print("CHART_CANDLES");break;
    default:Print("CHART_LINE");
}
bool shifted=ChartGetInteger(0,CHART_SHIFT);
if(shifted)
    Print("CHART_SHIFT = true");
else
    Print("CHART_SHIFT = false");
bool autoscroll=ChartGetInteger(0,CHART_AUTOSCROLL);
if(autoscroll)
    Print("CHART_AUTOSCROLL = true");
else
    Print("CHART_AUTOSCROLL = false");
int chartHandle=ChartGetInteger(0,CHART_WINDOW_HANDLE);

```

```
Print("CHART_WINDOW_HANDLE = ",chartHandle);
int windows=ChartGetInteger(0,CHART_WINDOWS_TOTAL);
Print("CHART_WINDOWS_TOTAL = ",windows);
if(windows>1)
{
    for(int i=0;i<windows;i++)
    {
        int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,i);
        double priceMin=ChartGetDouble(0,CHART_PRICE_MIN,i);
        double priceMax=ChartGetDouble(0,CHART_PRICE_MAX,i);
        Print(i+": CHART_HEIGHT_IN_PIXELS = ",height,"pixels");
        Print(i+": CHART_PRICE_MIN = ",priceMin);
        Print(i+": CHART_PRICE_MAX = ",priceMax);
    }
}
```

#### Смотри также

[Примеры работы с графиком](#)

## Позиционирование графика

Три идентификатора из списка ENUM\_CHART\_POSITION являются возможными значениями параметра position для функции [ChartNavigate\(\)](#).

### ENUM\_CHART\_POSITION

Идентификатор	Описание
CHART_BEGIN	Начало графика (самые старые цены)
CHART_CURRENT_POS	Текущая позиция
CHART_END	Конец графика (самые свежие цены)

**Пример:**

```
long handle=ChartOpen("EURUSD",PERIOD_H12);
if(handle!=0)
{
    ChartSetInteger(handle,CHART_AUTOSCROLL,false);
    ChartSetInteger(handle,CHART_SHIFT,true);
    ChartSetInteger(handle,CHART_MODE,CHART_LINE);
    ResetLastError();
    bool res=ChartNavigate(handle,CHART_END,150);
    if(!res)
        Print("Navigate failed. Error = ",GetLastError());
    ChartRedraw();
}
```

## Отображение графиков

Ценовые графики можно отображать тремя способами:

- в виде баров;
- в виде свечей;
- в виде ломаной линии.

Конкретный способ отображения ценового графика задается функцией [ChartSetInteger\(handle\\_графика, CHART\\_MODE, тип\\_графика\)](#), где тип\_графика - одно из значений перечисления ENUM\_CHART\_MODE.

### ENUM\_CHART\_MODE

Идентификатор	Описание
CHART_BARS	Отображение в виде баров
CHART_CANDLES	Отображение в виде японских свечей
CHART_LINE	Отображение в виде линии, проведенной по ценам Close

Для указания режима отображения объемов на ценовом графике используется функция [ChartSetInteger\(handle\\_графика, CHART\\_SHOW\\_VOLUMES, тип\\_отображения\)](#), где тип\_отображения - одно из значений перечисления ENUM\_CHART\_VOLUME\_MODE.

### ENUM\_CHART\_VOLUME\_MODE

Идентификатор	Описание
CHART_VOLUME_HIDE	Объемы не показаны
CHART_VOLUME_TICK	Тиковые объемы
CHART_VOLUME_REAL	Торговые объемы

**Пример:**

```
//--- получим handle текущего графика
long handle=ChartID();
if(handle>0) // если получилось, дополнительно настроим
{
    //--- отключим автопрокрутку
    ChartSetInteger(handle,CHART_AUTOSCROLL,false);
    //--- установим отступ правого края графика
    ChartSetInteger(handle,CHART_SHIFT,true);
    //--- отобразим в виде свечей
    ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
    //--- прокрутим на 100 баров от начала истории
    ChartNavigate(handle,CHART_CURRENT_POS,100);
    //--- установить режим отображения тиковых объемов
    ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);
}
```

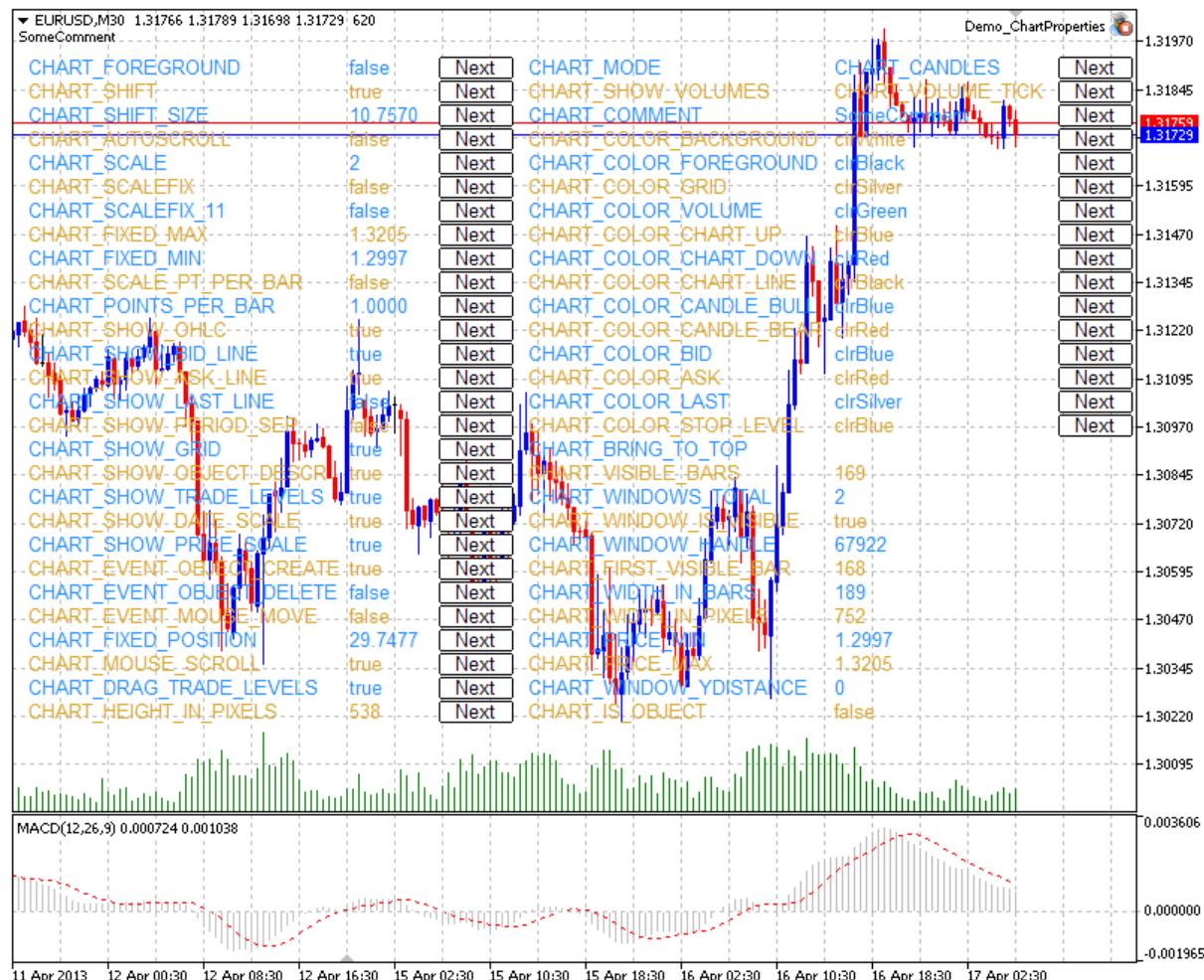
Смотри также

[ChartOpen](#), [ChartID](#)

## Примеры работы с графиком

В этом разделе представлены примеры для работы со свойствами графика. Для каждого свойства приведены одна или две законченные функции, которые позволяют задавать/получать значение этого свойства. Эти функции можно использовать в своих MQL5 программах как есть.

На рисунке показана графическая панель для наглядной демонстрации того, как изменение [свойства графика](#) меняет его внешний вид. Нажатие кнопки "Next" позволяет установить новое значение соответствующего свойства и увидеть изменения в окне графика.



Исходный код панели находится [ниже](#).

## Свойства графика и примеры функций для работы с ними

- **CHART\_IS\_OBJECT** - определяет, является ли объект настоящим графиком или [графическим объектом](#).

```
//+-----+
//| Определение того, является ли объект графиком. Если это
//| графический объект, то результат true. Если же это настоящий
//| график, то переменная result примет значение false.
//+-----+
bool ChartIsObject(bool &result,const long chart_ID=0)
```

```

{
//--- подготовим переменную для получения значения свойства
    long value;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим свойство графика
    if(!ChartGetInteger(chart_ID,CHART_IS_OBJECT,0,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        //--- вернем false
        return(false);
    }
//--- запомним в переменную значение свойства графика
    result=value;
//--- успешное выполнение
    return(true);
}

```

- **CHART\_BRING\_TO\_TOP** - показывает график поверх всех других.

```

//+-----+
//| Отправка терминалу команды на показ графика поверх всех других. |
//+-----+
bool ChartBringToTop(const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- покажем график поверх всех других
    if(!ChartSetInteger(chart_ID,CHART_BRING_TO_TOP,0,true))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_MOUSE\_SCROLL** - свойство прокрутки графика левой кнопкой мыши.

```

//+-----+
//| Функция определяет, можно ли прокручивать график левой кнопкой |
//| мыши. |
//+-----+
bool ChartMouseScrollGet(bool &result,const long chart_ID=0)
{

```

```

//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ", GetLastError());
    return(false);
}
//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

//+-----+
//| Функция включает/выключает режим прокрутки графика левой кнопкой |
//| мыши. |
//+-----+
bool ChartMouseScrollSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ", GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_EVENT\_MOUSE\_MOVE** - свойство отправки mql5-программам сообщений о событиях перемещения и нажатия кнопок мыши ([CHARTEVENT\\_MOUSE\\_MOVE](#)).

```

//+-----+
//| Отправляются ли сообщения о событиях перемещения и нажатия кнопок |
//| мыши всем mql5-программам на данном графике? |
//+-----+
bool ChartEventMouseMoveGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();

```

```

//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}

//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

//+-----+
//| Функция включает/выключает режим отправки сообщений о событиях |
//| перемещения и нажатия кнопок мыши всем mq5-программам на данном | |
//| графике. |
//+-----+
bool ChartEventMouseMoveSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

```

- **CHART\_EVENT\_OBJECT\_CREATE** - свойство отправки mq5-программам сообщений о событии создания графического объекта ([CHARTEVENT\\_OBJECT\\_CREATE](#)).

```

//+-----+
//| Отправляются ли сообщения о событии создания графического объекта|
//| всем mq5-программам на данном графике? |
//+-----+
bool ChartEventObjectCreateGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
{

```

```

//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

//+-----+
//| Функция включает/выключает режим отправки сообщений о событии |
//| создания графического объекта всем mq5-программам на данном      |
//| графике.                                                               |
//+-----+
bool ChartEventObjectCreateSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();

//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

```

- **CHART\_EVENT\_OBJECT\_DELETE** - свойство отправки mq5-программам сообщений о событии удаления графического объекта ([CHARTEVENT\\_OBJECT\\_DELETE](#)).

```

//+-----+
//| Отправляются ли сообщения о событии удаления графического объекта|
//| всем mq5-программам на данном графике?                                |
//+-----+
bool ChartEventObjectDeleteGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();

//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

```

```

    }

//--- запомним в переменную значение свойства графика
    result=value;
//--- успешное выполнение
    return(true);
}

//+-----+
//| Функция включает/выключает режим отправки сообщений о событии
//| удаления графического объекта всем mq5-программам на данном
//| графике.
//+-----+
bool ChartEventObjectDeleteSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- установим значение свойства
    if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_MODE** - тип графика (свечи, бары или линия).

```

//+-----+
//| Получение типа отображения графика (в виде свечей, баров или
//| линии).
//+-----+
ENUM_CHART_MODE ChartModeGet(const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long result=WRONG_VALUE;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_MODE,0,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((ENUM_CHART_MODE)result);
}
//+-----+

```

```
//| Установка типа отображения графика (в виде свечей, баров или
//| линии).
//+-----+
bool ChartModeSet(const long value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_MODE,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}
//--- успешное выполнение
return(true);
}
```

- **CHART\_FOREGROUND** - свойство отображения ценового графика на переднем плане.

```
//+-----+
//| Функция определяет, отображается ли ценовой график на переднем
//| плане.
//+-----+
bool ChartForegroundGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_FOREGROUND,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}
//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}
//+-----+
//| Функция включает/выключает режим отображения ценового графика на
//| переднем плане.
//+-----+
bool ChartForegroundSet(const bool value,const long chart_ID=0)
{
```

```

//--- сбросим значение ошибки
ResetLastError();

//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_FOREGROUND,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

```

- **CHART\_SHIFT** - режим отступа ценового графика от правого края.

```

//+-----+
//| Функция определяет, включен ли режим отображения ценового графика |
//| с отступом от правого края. |
//+-----+
bool ChartShiftGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_SHIFT,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}

//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

//+-----+
//| Функция включает/выключает режим отображения ценового графика с |
//| отступом от правого края. |
//+-----+
bool ChartShiftSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_SHIFT,0,value))
{
}

```

```

//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_AUTOSCROLL** - режим автоматического перехода к правому краю графика.

```

//+-----+
//| Функция определяет, включен ли режим автоматической прокрутки |
//| графика вправо при поступлении новых тиков. |
//+-----+
bool ChartAutoscrollGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}
//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}
//+-----+
//| Функция включает/выключает режим автоматической прокрутки графика|
//| вправо при поступлении новых тиков. |
//+-----+
bool ChartAutoscrollSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}
//--- успешное выполнение

```

```

    return(true);
}

```

- **CHART\_SCALE** - свойство масштаба графика.

```

//+-----+
//| Получение масштаба графика (от 0 до 5). |
//+-----+
int ChartScaleGet(const long chart_ID=0)
{
//--- подготавим переменную для получения значения свойства
    long result=-1;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_SCALE,0,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((int)result);
}
//+-----+
//| Установка масштаба графика (от 0 до 5). |
//+-----+
bool ChartScaleSet(const long value,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- установим значение свойства
    if(!ChartSetInteger(chart_ID,CHART_SCALE,0,value))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_SCALEFIX** - режим фиксированного масштаба графика.

```

//+-----+
//| Функция определяет, включен ли режим фиксированного масштаба. |
//+-----+
bool ChartScaleFixGet(bool &result,const long chart_ID=0)

```

```

{
//--- подготовим переменную для получения значения свойства
    long value;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_SCALEFIX,0,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- запомним в переменную значение свойства графика
    result=value;
//--- успешное выполнение
    return(true);
}
//----------------------------------------------------------------------------------------------------------------+
//| Функция включает/выключает режим фиксированного масштаба. |
//----------------------------------------------------------------------------------------------------------------+
bool ChartScaleFixSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- установим значение свойства
    if(!ChartSetInteger(chart_ID,CHART_SCALEFIX,0,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_SCALEFIX\_11** - режим масштаба графика 1:1.

```

//----------------------------------------------------------------------------------------------------------------+
//| Функция определяет, включен ли режим масштаба "1:1". |
//----------------------------------------------------------------------------------------------------------------+
bool ChartScaleFix11Get(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long value;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_SCALEFIX_11,0,value))

```

```

{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ", GetLastError());
    return(false);
}

//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

//+-----+
// | Функция включает/выключает режим масштаба "1:1"           |
//+-----+
bool ChartScaleFix11Set(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ", GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_SCALE\_PT\_PER\_BAR** - режим указания масштаба графика в пунктах на бар.

```

//+-----+
// | Функция определяет, включен ли режим указания масштаба в пунктах |
// | на бар.                                                               |
//+-----+
bool ChartScalePerBarGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ", GetLastError());
    return(false);
}
//--- запомним в переменную значение свойства графика

```

```

    result=value;
//--- успешное выполнение
    return(true);
}
//+-----+
//| Функция включает/выключает режим указания масштаба в пунктах на |
//| бар. |
//+-----+
bool ChartScalePerBarSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- установим значение свойства
    if(!ChartSetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_SHOW\_OHLC** - свойство отображения в левом верхнем углу значений OHLC.

```

//+-----+
//| Функция определяет, включен ли режим отображения значений OHLC |
//| в левом верхнем углу. |
//+-----+
bool ChartShowOHLCGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long value;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- запомним в переменную значение свойства графика
    result=value;
//--- успешное выполнение
    return(true);
}
//+-----+

```

```

//| Функция включает/выключает режим отображения значений OHLC в |
//| левом верхнем углу графика. |
//+-----+
bool ChartShowOHLCSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_SHOW\_BID\_LINE** - свойство отображения значения Bid горизонтальной линией на графике.

```

//+-----+
//| Функция определяет, включен ли режим отображения линии Bid на |
//| графике. |
//+-----+
bool ChartShowBidLineGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_SHOW_BID_LINE,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}
//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}
//+-----+
//| Функция включает/выключает режим отображения линии Bid на |
//| графике. |
//+-----+
bool ChartShowBidLineSet(const bool value,const long chart_ID=0)

```

```

{
//--- сбросим значение ошибки
ResetLastError();

//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_SHOW_BID_LINE,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

```

- **CHART\_SHOW\_ASK\_LINE** - свойство отображения значения Ask горизонтальной линией на графике.

```

//+-----+
//| Функция определяет, включен ли режим отображения линии Ask на
//| графике.
//+-----+
bool ChartShowAskLineGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}
//+-----+
//| Функция включает/выключает режим отображения линии Ask на
//| графике.
//+-----+
bool ChartShowAskLineSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства

```

```

if(!ChartSetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_SHOW\_LAST\_LINE** - свойство отображения значения Last горизонтальной линией на графике.

```

//+-----+
//| Функция определяет, включен ли режим отображения линии для цены |
//| последней совершенной сделки. |
//+-----+
bool ChartShowLastLineGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}
//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}
//+-----+
//| Функция включает/выключает режим отображения линии цены последней|
//| совершенной сделки. |
//+-----+
bool ChartShowLastLineSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
}

```

```

        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_SHOW\_PERIOD\_SEP** - свойство отображения вертикальных разделителей между соседними периодами.

```

//+-----+
//| Функция определяет, включен ли режим отображения вертикальных |
//| разделителей между соседними периодами. |
//+-----+
bool ChartShowPeriodSeparatorGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long value;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- запомним в переменную значение свойства графика
    result=value;
//--- успешное выполнение
    return(true);
}
//+-----+
//| Функция включает/выключает режим отображения вертикальных |
//| разделителей между соседними периодами. |
//+-----+
bool ChartShowPeriodSepapatorSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- установим значение свойства
    if(!ChartSetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

```
}
```

- **CHART\_SHOW\_GRID** - свойство отображения сетки на графике.

```
//+-----+
//| Функция определяет, отображается ли сетка на графике. |
//+-----+
bool ChartShowGridGet(bool &result,const long chart_ID=0)
{
    //--- подготовим переменную для получения значения свойства
    long value;
    //--- сбросим значение ошибки
    ResetLastError();
    //--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_SHOW_GRID,0,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
    //--- запомним в переменную значение свойства графика
    result=value;
    //--- успешное выполнение
    return(true);
}
//+-----+
//| Функция включает/выключает отображение сетки на графике. |
//+-----+
bool ChartShowGridSet(const bool value,const long chart_ID=0)
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- установим значение свойства
    if(!ChartSetInteger(chart_ID,CHART_SHOW_GRID,0,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}
```

- **CHART\_SHOW\_VOLUMES** - свойство отображения объемов на графике.

```
//+-----+
//| Функция определяет, отображаются ли объемы на графике (не |
//+-----+
```

```

//| отображаются, отображаются тиковые, отображаются реальные).
//+-----+
ENUM_CHART_VOLUME_MODE ChartShowVolumesGet(const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long result=WRONG_VALUE;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_SHOW_VOLUMES,0,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ", GetLastError());
    }
//--- вернем значение свойства графика
    return((ENUM_CHART_VOLUME_MODE)result);
}
//+-----+
//| Функция устанавливает режим отображения объемов на графике.
//+-----+
bool ChartShowVolumesSet(const long value,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- установим значение свойства
    if(!ChartSetInteger(chart_ID,CHART_SHOW_VOLUMES,value))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ", GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_SHOW\_OBJECT\_DESCR** - свойство всплывающих описаний графических объектов.

```

//+-----+
//| Функция определяет, отображаются ли всплывающие описания
//| графических объектов при наведение на них мышью.
//+-----+
bool ChartShowObjectDescriptionGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long value;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства

```

```

if(!ChartGetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}

//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

//+-----+
//| Функция включает/выключает режим отображения всплывающих описаний|
//| графических объектов при наведение на них мышью. |
//+-----+
bool ChartShowObjectDescriptionSet(const bool value,const long chart_ID=0)
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- установим значение свойства
    if(!ChartSetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }

    //--- успешное выполнение
    return(true);
}

```

- **CHART\_VISIBLE\_BARS** - определяет количество баров на графике, доступных для отображения.

```

//+-----+
//| Функция получает количество баров, которые отображаются (видимы ) |
//| в окне графика. |
//+-----+
int ChartVisibleBars(const long chart_ID=0)
{
    //--- подготовим переменную для получения значения свойства
    long result=-1;
    //--- сбросим значение ошибки
    ResetLastError();
    //--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_VISIBLE_BARS,0,result))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
}

```

```
//--- вернем значение свойства графика
    return((int)result);
}
```

- **CHART\_WINDOWS\_TOTAL** - определяет общее количество окон графика, включая подокна индикаторов.

```
//+-----+
//| Функция получает общее количество окон графика, включая подокна |
//| индикаторов. |
//+-----+
int ChartWindowsTotal(const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long result=-1;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_WINDOWS_TOTAL,0,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((int)result);
}
```

- **CHART\_WINDOW\_IS\_VISIBLE** - определяет видимость подокна.

```
//+-----+
//| Функция определяет, является ли данное окно или подокно графика |
//| видимым. |
//+-----+
bool ChartWindowsIsVisible(bool &result,const long chart_ID=0,const int sub_window=0)
{
//--- подготовим переменную для получения значения свойства
    long value;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_WINDOW_IS_VISIBLE,sub_window,value))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- запомним в переменную значение свойства графика
```

```

    result=value;
//--- успешное выполнение
    return(true);
}

```

- **CHART\_WINDOW\_HANDLE** - возвращает хэндл графика.

```

//+-----+
// | Функция получает хэндл графика
//+-----+
int ChartWindowsHandle(const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long result=-1;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_WINDOW_HANDLE,0,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((int)result);
}

```

- **CHART\_WINDOW\_YDISTANCE** - определяет дистанцию в пикселях между верхней рамкой подокна индикатора и верхней рамкой главного окна графика.

```

//+-----+
// | Функция получает расстояние в пикселях между верхней рамкой
// | подокна и верхней рамкой главного окна графика.
//+-----+
int ChartWindowsYDistance(const long chart_ID=0,const int sub_window=0)
{
//--- подготовим переменную для получения значения свойства
    long result=-1;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_WINDOW_YDISTANCE,sub_window,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((int)result);
}

```

```
}
```

- **CHART\_FIRST\_VISIBLE\_BAR** - возвращает номер первого видимого бара на графике (индексация баров соответствует [таймсерии](#)).

```
//+-----+
//| Функция получает номер первого видимого бара на графике.           |
//| Индексация как в таймсерии, последние бары имеют меньшие индексы.   |
//+-----+
int ChartFirstVisibleBar(const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long result=-1;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_FIRST_VISIBLE_BAR,0,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((int)result);
}
```

- **CHART\_WIDTH\_IN\_BARS** - возвращает ширину графика в барах.

```
//+-----+
//| Функция получает значение ширины графика в барах.                   |
//+-----+
int ChartWidthInBars(const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long result=-1;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_BARS,0,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((int)result);
}
```

- **CHART\_WIDTH\_IN\_PIXELS** - возвращает ширину графика в пикселях.

```

//+-----+
//| Функция получает значение ширины графика в пикселях. |
//+-----+
int ChartWidthInPixels(const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long result=-1;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_PIXELS,0,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((int)result);
}

```

- **CHART\_HEIGHT\_IN\_PIXELS** - свойство высоты графика в пикселях.

```

//+-----+
//| Функция получает значение высоты графика в пикселях. |
//+-----+
int ChartHeightInPixelsGet(const long chart_ID=0,const int sub_window=0)
{
//--- подготовим переменную для получения значения свойства
    long result=-1;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((int)result);
}
//+-----+
//| Функция устанавливает значение высоты графика в пикселях. |
//+-----+
bool ChartHeightInPixelsSet(const int value,const long chart_ID=0,const int sub_window)
{
//--- сбросим значение ошибки
    ResetLastError();
}

```

```

//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_COLOR\_BACKGROUND** - цвет фона графика.

```

//+-----+
//| Функция получает цвет фона графика. |
//+-----+
color ChartBackColorGet(const long chart_ID=0)
{
//--- подготовим переменную для получения цвета
long result=clrNONE;
//--- сбросим значение ошибки
ResetLastError();
//--- получим цвет фона графика
if(!ChartGetInteger(chart_ID,CHART_COLOR_BACKGROUND,0,result))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
}
//--- вернем значение свойства графика
return((color)result);
}
//+-----+
//| Функция устанавливает цвет фона графика. |
//+-----+
bool ChartBackColorSet(const color clr,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим цвет фона графика
if(!ChartSetInteger(chart_ID,CHART_COLOR_BACKGROUND,clr))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_COLOR\_FOREGROUND** - цвет осей, шкалы и строки OHLC.

```
//+-----+
//| Функция получает цвет осей, шкалы и строки OHLC графика.      |
//+-----+
color ChartForeColorGet(const long chart_ID=0)
{
    //--- подготовим переменную для получения цвета
    long result=clrNONE;
    //--- сбросим значение ошибки
    ResetLastError();
    //--- получим цвет осей, шкалы и строки OHLC графика
    if(!ChartGetInteger(chart_ID,CHART_COLOR_FOREGROUND,0,result))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ", GetLastError());
    }
    //--- вернем значение свойства графика
    return((color)result);
}
//+-----+
//| Функция устанавливает цвет осей, шкалы и строки OHLC графика.      |
//+-----+
bool ChartForeColorSet(const color clr,const long chart_ID=0)
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- установим цвет осей, шкалы и строки OHLC графика
    if(!ChartSetInteger(chart_ID,CHART_COLOR_FOREGROUND,clr))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ", GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}
```

- **CHART\_COLOR\_GRID** - цвет сетки графика.

```
//+-----+
//| Функция получает цвет сетки графика.      |
//+-----+
color ChartGridColorGet(const long chart_ID=0)
{
    //--- подготовим переменную для получения цвета
```

```

long result=clrNONE;
//--- сбросим значение ошибки
ResetLastError();
//--- получим цвет сетки графика
if(!ChartGetInteger(chart_ID,CHART_COLOR_GRID,0,result))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
}
//--- вернем значение свойства графика
return((color)result);
}

//+-----+
//| Функция устанавливает цвет сетки графика. |
//+-----+
bool ChartGridColorSet(const color clr,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим цвет сетки графика
if(!ChartSetInteger(chart_ID,CHART_COLOR_GRID,clr))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_COLOR\_VOLUME** - цвет объемов и уровней открытия позиций.

```

//+-----+
//| Функция получает цвет отображения объемов и уровней открытия |
//| позиций. |
//+-----+
color ChartVolumeColorGet(const long chart_ID=0)
{
//--- подготовим переменную для получения цвета
long result=clrNONE;
//--- сбросим значение ошибки
ResetLastError();
//--- получим цвет объемов и уровней открытия позиций
if(!ChartGetInteger(chart_ID,CHART_COLOR_VOLUME,0,result))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
}

```

```

//--- вернем значение свойства графика
    return((color)result);
}
//+-----+
//| Функция устанавливает цвет отображения объемов и уровней открытия|
//| позиций.|
//+-----+
bool ChartVolumeColorSet(const color clr,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- установим цвет объемов и уровней открытия позиций
    if(!ChartSetInteger(chart_ID,CHART_COLOR_VOLUME,clr))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_COLOR\_CHART\_UP** - цвет бара вверх, тени и окантовки тела бычьей свечи.

```

//+-----+
//| Функция получает цвет бара направленного вверх, цвет тени и |
//| окантовки тела бычьей свечи.|
//+-----+
color ChartUpColorGet(const long chart_ID=0)
{
//--- подготовим переменную для получения цвета
    long result=clrNONE;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим цвет бара вверх, тени и окантовки тела бычьей свечи
    if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_UP,0,result))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((color)result);
}
//+-----+
//| Функция устанавливает цвет бара направленного вверх, цвет тени и |
//| окантовки тела бычьей свечи.|
//+-----+
bool ChartUpColorSet(const color clr,const long chart_ID=0)

```

```

{
//--- сбросим значение ошибки
ResetLastError();

//--- установим цвет бара вверх, тени и окантовки тела бычьей свечи
if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_UP,clr))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

```

- **CHART\_COLOR\_CHART\_DOWN** - цвет бара вниз, тени и окантовки тела медвежьей свечи.

```

//+-----+
//| Функция получает цвет бара направленного вниз, цвет тени и |
//| окантовки тела медвежьей свечи. |
//+-----+
color ChartDownColorGet(const long chart_ID=0)
{
//--- подготовим переменную для получения цвета
long result=clrNONE;
//--- сбросим значение ошибки
ResetLastError();

//--- получим цвет бара вниз, тени и окантовки тела медвежьей свечи
if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_DOWN,0,result))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
}

//--- вернем значение свойства графика
return((color)result);
}

//+-----+
//| Функция устанавливает цвет бара направленного вниз, цвет тени и |
//| окантовки тела медвежьей свечи. |
//+-----+
bool ChartDownColorSet(const color clr,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();

//--- установим цвет бара вниз, тени и окантовки тела медвежьей свечи
if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_DOWN,clr))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
}
}

```

```

        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_COLOR\_CHART\_LINE** - цвет линии графика и японских свечей "Доджи".

```

//+-----+
//| Функция получает цвет линии графика и японских свечей "Доджи". |
//+-----+
color ChartLineColorGet(const long chart_ID=0)
{
//--- подготовим переменную для получения цвета
    long result=clrNONE;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим цвет линии графика и японских свечей "Доджи"
    if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_LINE,0,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((color)result);
}
//+-----+
//| Функция устанавливает цвет линии графика и японских свечей |
//| "Доджи". |
//+-----+
bool ChartLineColorSet(const color clr,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- установим цвет линии графика и японских свечей "Доджи"
    if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_LINE,clr))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_COLOR\_CANDLE\_BULL** - цвет тела бычьей свечи.

```

//+-----+
//| Функция получает цвет тела бычьей свечи. |
//+-----+
color ChartBullColorGet(const long chart_ID=0)
{
    //--- подготовим переменную для получения цвета
    long result=clrNONE;
    //--- сбросим значение ошибки
    ResetLastError();
    //--- получим цвет тела бычьей свечи
    if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,0,result))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
    //--- вернем значение свойства графика
    return((color)result);
}
//+-----+
//| Функция устанавливает цвет тела бычьей свечи. |
//+-----+
bool ChartBullColorSet(const color clr,const long chart_ID=0)
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- установим цвет тела бычьей свечи
    if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,clr))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

```

- **CHART\_COLOR\_CANDLE\_BEAR** - цвет тела медвежьей свечи.

```

//+-----+
//| Функция получает цвет тела медвежьей свечи. |
//+-----+
color ChartBearColorGet(const long chart_ID=0)
{
    //--- подготовим переменную для получения цвета
    long result=clrNONE;
    //--- сбросим значение ошибки
    ResetLastError();
    //--- получим цвет тела медвежьей свечи

```

```

if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,0,result))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
}

//--- вернем значение свойства графика
return((color)result);
}

//+-----+
// | Функция устанавливает цвет тела медвежьей свечи. |
//+-----+

bool ChartBearColorSet(const color clr,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();

//--- установим цвет тела медвежьей свечи
if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,clr))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

```

- **CHART\_COLOR\_BID** - цвет линии цены Bid.

```

//+-----+
// | Функция получает цвет отображения линии Bid. |
//+-----+

color ChartBidColorGet(const long chart_ID=0)
{
//--- подготовим переменную для получения цвета
long result=clrNONE;

//--- сбросим значение ошибки
ResetLastError();

//--- получим цвет линии цены Bid
if(!ChartGetInteger(chart_ID,CHART_COLOR_BID,0,result))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
}

//--- вернем значение свойства графика
return((color)result);
}

//+-----+
// | Функция устанавливает цвет отображения линии Bid. |

```

```

//+-----+
bool ChartBidColorSet(const color clr,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим цвет линии цены Bid
if(!ChartSetInteger(chart_ID,CHART_COLOR_BID,clr))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_COLOR\_ASK** - цвет линии цены Ask.

```

//+-----+
//| Функция получает цвет отображения линии Ask. |
//+-----+
color ChartAskColorGet(const long chart_ID=0)
{
//--- подготовим переменную для получения цвета
long result=clrNONE;
//--- сбросим значение ошибки
ResetLastError();
//--- получим цвет линии цены Ask
if(!ChartGetInteger(chart_ID,CHART_COLOR_ASK,0,result))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
}
//--- вернем значение свойства графика
return((color)result);
}
//+-----+
//| Функция устанавливает цвет отображения линии Ask. |
//+-----+
bool ChartAskColorSet(const color clr,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим цвет линии цены Ask
if(!ChartSetInteger(chart_ID,CHART_COLOR_ASK,clr))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
}
}

```

```

        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_COLOR\_LAST** - цвет линии цены последней совершенной сделки (Last).

```

//+-----+
//| Функция получает цвет линии цены последней совершенной сделки. |
//+-----+
color ChartLastColorGet(const long chart_ID=0)
{
//--- подготовим переменную для получения цвета
    long result=clrNONE;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим цвет линии цены последней совершенной сделки (Last)
    if(!ChartGetInteger(chart_ID,CHART_COLOR_LAST,0,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return((color)result);
}
//+-----+
//| Функция устанавливает цвет линии цены последней совершенной |
//| сделки. |
//+-----+
bool ChartLastColorSet(const color clr,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- установим цвет линии цены последней совершенной сделки (Last)
    if(!ChartSetInteger(chart_ID,CHART_COLOR_LAST,clr))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

```

- **CHART\_COLOR\_STOP\_LEVEL** - цвет уровней стоп-ордеров (Stop Loss и Take Profit).

```

//+-----+
//| Функция получает цвета уровней Stop Loss и Take Profit.      |
//+-----+
color ChartStopLevelColorGet(const long chart_ID=0)
{
    //--- подготовим переменную для получения цвета
    long result=clrNONE;
    //--- сбросим значение ошибки
    ResetLastError();
    //--- получим цвет уровней стоп-ордеров (Stop Loss и Take Profit)
    if(!ChartGetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,0,result))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
    //--- вернем значение свойства графика
    return((color)result);
}
//+-----+
//| Функция устанавливает цвета уровней Stop Loss и Take Profit.      |
//+-----+
bool ChartStopLevelColorSet(const color clr,const long chart_ID=0)
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- установим цвет уровней стоп-ордеров (Stop Loss и Take Profit)
    if(!ChartSetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,clr))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

```

- **CHART\_SHOW\_TRADE\_LEVELS** - свойство отображения на графике торговых уровней (уровни открытых позиций, Stop Loss, Take Profit и отложенных ордеров).

```

//+-----+
//| Функция определяет, отображаются ли на графике торговые уровни.      |
//+-----+
bool ChartShowTradeLevelsGet(bool &result,const long chart_ID=0)
{
    //--- подготовим переменную для получения значения свойства
    long value;
    //--- сбросим значение ошибки
    ResetLastError();

```

```

//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_SHOW_TRADE_LEVELS,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}

//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

//+-----+
//| Функция включает/выключает режим отображения торговых уровней. |
//+-----+
bool ChartShowTradeLevelsSet(const bool value,const long chart_ID=0)
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- установим значение свойства
    if(!ChartSetInteger(chart_ID,CHART_SHOW_TRADE_LEVELS,0,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }

    //--- успешное выполнение
    return(true);
}

```

- **CHART\_DRAG\_TRADE\_LEVELS** - свойство разрешения на перетаскивание торговых уровней на графике с помощью мышки.

```

//+-----+
//| Функция определяет, можно ли перетаскивать торговые уровни на |
//| графике при помощи мыши. |
//+-----+
bool ChartDragTradeLevelsGet(bool &result,const long chart_ID=0)
{
    //--- подготовим переменную для получения значения свойства
    long value;
    //--- сбросим значение ошибки
    ResetLastError();
    //--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
}

```

```

        return(false);
    }

//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

//+-----+
//| Функция включает/выключает режим перетаскивания торговых уровней |
//| на графике при помощи мыши. |
//+-----+

bool ChartDragTradeLevelsSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();

//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

```

- **CHART\_SHOW\_DATE\_SCALE** - свойство отображения на графике шкалы времени.

```

//+-----+
//| Функция определяет, отображается ли на графике шкала времени. |
//+-----+

bool ChartShowDateScaleGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();

//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

```

```

    }

//+-----+
//| Функция включает/выключает режим отображения шкалы времени на |
//| графике. |
//+-----+

bool ChartShowDateScaleSet(const bool value,const long chart_ID=0)
{
    //--- сбросим значение ошибки
    ResetLastError();

    //--- установим значение свойства
    if(!ChartSetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }

    //--- успешное выполнение
    return(true);
}

```

- **CHART\_SHOW\_PRICE\_SCALE** - свойство отображения на графике шкалы цены.

```

//+-----+
//| Функция определяет, отображается ли на графике шкала цены. |
//+-----+

bool ChartShowPriceScaleGet(bool &result,const long chart_ID=0)
{
    //--- подготовим переменную для получения значения свойства
    long value;
    //--- сбросим значение ошибки
    ResetLastError();

    //--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }

    //--- запомним в переменную значение свойства графика
    result=value;
    //--- успешное выполнение
    return(true);
}

//+-----+
//| Функция включает/выключает режим отображения шкалы цены на |
//| графике. |
//+-----+

bool ChartShowPriceScaleSet(const bool value,const long chart_ID=0)

```

```

{
//--- сбросим значение ошибки
ResetLastError();

//--- установим значение свойства
if(!ChartSetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

```

- **CHART\_SHOW\_ONE\_CLICK** - отображение на графике панели быстрой торговли (опция "Торговля одним кликом").

```

//+-----+
//| Функция определяет, отображается ли на графике
//| панель быстрой торговли ("Торговля одним кликом")
//+-----+
bool ChartShowOneClickPanelGet(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_SHOW_ONE_CLICK,0,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

//+-----+
//| Функция включает/выключает режим отображения
//| панели быстрой торговли на графике
//+-----+
bool ChartShowOneClickPanelSet(const bool value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства

```

```

if(!ChartSetInteger(chart_ID,CHART_SHOW_ONE_CLICK,0,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_SHIFT\_SIZE** - размер отступа нулевого бара от правого края в процентах.

```

//+-----+
//| Функция получает размер отступа нулевого бара от правого края |
//| графика в процентах (от 10% до 50%). |
//+-----+
double ChartShiftSizeGet(const long chart_ID=0)
{
//--- подготовим переменную для получения результата
    double result=EMPTY_VALUE;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetDouble(chart_ID,CHART_SHIFT_SIZE,0,result))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return(result);
}
//+-----+
//| Функция устанавливает размер отступа нулевого бара от правого |
//| края графика в процентах (от 10% до 50%). Для включения режима |
//| отступа, нужно установить значение свойства CHART_SHIFT равным |
//| true. |
//+-----+
bool ChartShiftSizeSet(const double value,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- установим значение свойства
    if(!ChartSetDouble(chart_ID,CHART_SHIFT_SIZE,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
}

```

```
//--- успешное выполнение
    return(true);
}
```

- **CHART\_FIXED\_POSITION** - положение фиксированной позиции графика от левого края в процентах.

```
//----------------------------------------------------------------------------------------------------------------+
//| Функция получает положение фиксированной позиции графика от |
//| левого края в процентах. |
//----------------------------------------------------------------------------------------------------------------+
double ChartFixedPositionGet(const long chart_ID=0)
{
    //--- подготовим переменную для получения результата
    double result=EMPTY_VALUE;
    //--- сбросим значение ошибки
    ResetLastError();
    //--- получим значение свойства
    if(!ChartGetDouble(chart_ID,CHART_FIXED_POSITION,0,result))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
    //--- вернем значение свойства графика
    return(result);
}
//----------------------------------------------------------------------------------------------------------------+
//| Функция устанавливает положение фиксированной позиции графика от |
//| левого края в процентах. Для того чтобы увидеть положение |
//| фиксированной позиции на графике, нужно предварительно задать |
//| значение свойства CHART_AUTOSCROLL равным false. |
//----------------------------------------------------------------------------------------------------------------+
bool ChartFixedPositionSet(const double value,const long chart_ID=0)
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- установим значение свойства
    if(!ChartSetDouble(chart_ID,CHART_FIXED_POSITION,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}
```

- **CHART\_FIXED\_MAX** - свойство фиксированного максимума графика.

```

//+-----+
//| Функция получает значение фиксированного максимума графика. |
//+-----+
double ChartFixedMaxGet(const long chart_ID=0)
{
    //--- подготовим переменную для получения результата
    double result=EMPTY_VALUE;
    //--- сбросим значение ошибки
    ResetLastError();
    //--- получим значение свойства
    if(!ChartGetDouble(chart_ID,CHART_FIXED_MAX,0,result))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
    //--- вернем значение свойства графика
    return(result);
}
//+-----+
//| Функция устанавливает значение фиксированного максимума графика. |
//| Для того чтобы удалось изменить значение этого свойства, нужно |
//| предварительно задать значение свойства CHART_SCALEFIX равным |
//| true. |
//+-----+
bool ChartFixedMaxSet(const double value,const long chart_ID=0)
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- установим значение свойства
    if(!ChartSetDouble(chart_ID,CHART_FIXED_MAX,value))
    {
        //--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

```

- **CHART\_FIXED\_MIN** - свойство фиксированного минимума графика.

```

//+-----+
//| Функция получает значение фиксированного минимума графика. |
//+-----+
double ChartFixedMinGet(const long chart_ID=0)
{

```

```

//--- подготовим переменную для получения результата
double result=EMPTY_VALUE;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetDouble(chart_ID,CHART_FIXED_MIN,0,result))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
}
//--- вернем значение свойства графика
return(result);
}

//+-----+
//| Функция устанавливает значение фиксированного минимума графика. |
//| Для того чтобы удалось изменить значение этого свойства, нужно |
//| предварительно задать значение свойства CHART_SCALEFIX равным |
//| true. |
//+-----+
bool ChartFixedMinSet(const double value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим значение свойства
if(!ChartSetDouble(chart_ID,CHART_FIXED_MIN,value))
{
    //--- выведем сообщение об ошибке в журнал "Эксперты"
    Print(__FUNCTION__+", Error Code = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

- **CHART\_POINTS\_PER\_BAR** - значение масштаба в пунктах на бар.

```

//+-----+
//| Функция получает значение масштаба графика в пунктах на бар. |
//+-----+
double ChartPointsPerBarGet(const long chart_ID=0)
{
//--- подготовим переменную для получения результата
double result=EMPTY_VALUE;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetDouble(chart_ID,CHART_POINTS_PER_BAR,0,result))
{

```

```

//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ", GetLastError());
}

//--- вернем значение свойства графика
return(result);
}

//+-----+
//| Функция устанавливает значение масштаба графика в пунктах на бар. |
//| Для того чтобы посмотреть результат изменения значения этого |
//| свойства, нужно предварительно задать значение свойства |
//| CHART_SCALE_PT_PER_BAR равным true. |
//+-----+
bool ChartPointsPerBarSet(const double value,const long chart_ID=0)
{
//--- сбросим значение ошибки
ResetLastError();

//--- установим значение свойства
if(!ChartSetDouble(chart_ID,CHART_POINTS_PER_BAR,value))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ", GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

```

- **CHART\_PRICE\_MIN** - возвращает значение минимума графика.

```

//+-----+
//| Функция получает значение минимума графика в главном окне или |
//| подокне. |
//+-----+
double ChartPriceMin(const long chart_ID=0,const int sub_window=0)
{
//--- подготовим переменную для получения результата
double result=EMPTY_VALUE;
//--- сбросим значение ошибки
ResetLastError();

//--- получим значение свойства
if(!ChartGetDouble(chart_ID,CHART_PRICE_MIN,sub_window,result))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ", GetLastError());
}

//--- вернем значение свойства графика
return(result);
}

```

- **CHART\_PRICE\_MAX** - возвращает значение максимума графика.

```

//+-----+
//| Функция получает значение максимума графика в главном окне или |
//| подокне. |
//+-----+
double ChartPriceMax(const long chart_ID=0,const int sub_window=0)
{
//--- подготовим переменную для получения результата
    double result=EMPTY_VALUE;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetDouble(chart_ID,CHART_PRICE_MAX,sub_window,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
    }
//--- вернем значение свойства графика
    return(result);
}

```

- **CHART\_COMMENT** - текст комментария на графике.

```

//+-----+
//| Функция получает текст комментария в левом верхнем углу графика. |
//+-----+
bool ChartCommentGet(string &result,const long chart_ID=0)
{
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetString(chart_ID,CHART_COMMENT,result))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Функция устанавливает текст комментария в левом верхнем углу |
//| графика. |
//+-----+
bool ChartCommentSet(const string str,const long chart_ID=0)

```

```

{
//--- сбросим значение ошибки
ResetLastError();

//--- установим значение свойства
if(!ChartSetString(chart_ID,CHART_COMMENT,str))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

```

- **CHART\_IS\_MAXIMIZED** - окно графика развернуто

```

+-----+
//| Функция определяет, является ли данное окно графика развернутым |
+-----+
bool ChartWindowsIsMaximized(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
long value;
//--- сбросим значение ошибки
ResetLastError();
//--- получим значение свойства
if(!ChartGetInteger(chart_ID,CHART_IS_MAXIMIZED))
{
//--- выведем сообщение об ошибке в журнал "Эксперты"
Print(__FUNCTION__+", Error Code = ",GetLastError());
return(false);
}
//--- запомним в переменную значение свойства графика
result=value;
//--- успешное выполнение
return(true);
}

```

- **CHART\_IS\_MINIMIZED** - окно графика свернуто

```

//+-----+
//| Функция определяет, является ли данное окно графика свернутым |
//+-----+
bool ChartWindowsIsMinimized(bool &result,const long chart_ID=0)
{
//--- подготовим переменную для получения значения свойства
    long value;
//--- сбросим значение ошибки
    ResetLastError();
//--- получим значение свойства
    if(!ChartGetInteger(chart_ID,CHART_IS_MINIMIZED))
    {
//--- выведем сообщение об ошибке в журнал "Эксперты"
        Print(__FUNCTION__+", Error Code = ",GetLastError());
        return(false);
    }
//--- запомним в переменную значение свойства графика
    result=value;
//--- успешное выполнение
    return(true);
}

```

### Панель для свойств графика

```

//--- подключим библиотеку элементов управления
#include <ChartObjects\ChartObjectsTxtControls.mqh>
//--- предопределенные константы
#define X_PROPERTY_NAME_1      10 // x-координата имени свойства в первом столбце
#define X_PROPERTY_VALUE_1     225 // x-координата значения свойства в первом столбце
#define X_PROPERTY_NAME_2      345 // x-координата имени свойства во втором и третьем столбцах
#define X_PROPERTY_VALUE_2     550 // x-координата значения свойства во втором и третьем столбцах
#define X_BUTTON_1              285 // x-координата кнопки в первом столбце
#define X_BUTTON_2              700 // x-координата кнопки во втором столбце
#define Y_PROPERTY_1              30 // y-координата начала первого и второго столбца
#define Y_PROPERTY_2              286 // y-координата начала третьего столбца
#define Y_DISTANCE                  16 // расстояние по оси y между строками
#define LAST_PROPERTY_NUMBER 111 // номер последнего графического свойства
//--- входные параметры
input color InpFirstColor=clrDodgerBlue; // Цвет нечетных строк
input color InpSecondColor=clrGoldenrod; // Цвет четных строк
//--- переменные и массивы
CChartObjectLabel ExtLabelsName[]; // надписи для отображения имен свойств
CChartObjectLabel ExtLabelsValue[]; // надписи для отображения значений свойств
CChartObjectButton ExtButtons[]; // кнопки
int ExtNumbers[]; // индексы свойств
string ExtNames[]; // имена свойств
uchar ExtDataTypes[]; // типы данных свойств (integer, double, string)
uint ExtGroupTypes[]; // массив, который хранит данные о принадлежности
uchar ExtDrawTypes[]; // массив, который хранит данные о способе отображения
double ExtMaxValue[]; // максимальные значения свойств, которые они могут принимать
double ExtMinValue[]; // минимальные значения свойств, которые они могут принимать

```

```

double          ExtStep[];           // шаги для изменений свойств
int             ExtCount;          // общее количество всех свойств
color           ExtColors[2];       // массив цветов для отображения строк
string          ExtComments[2];     // массив комментариев (для свойства CHART_COMMENT)
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- выводим комментарий на график
Comment("SomeComment");
//--- сохраним цвета в массив для дальнейшего переключения между ними
ExtColors[0]=InpFirstColor;
ExtColors[1]=InpSecondColor;
//--- сохраним комментарии в массив для дальнейшего переключения между ними
ExtComments[0]="FirstComment";
ExtComments[1]="SecondComment";
//--- подготовим и отобразим панель управления свойствами графика
if(!PrepareControls())
    return(INIT_FAILED);
//--- успешное выполнение
return(INIT_SUCCEEDED);
}
//+-----+
//| Deinitialization function of the expert           |
//+-----+
void OnDeinit(const int reason)
{
//--- очищаем текст комментария на графике
Comment("");
}
//+-----+
//| Обработчик событий графика                      |
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
//--- проверка события нажатия на объект графика
if(id==CHARTEVENT_OBJECT_CLICK)
{
//--- разобьем имя объекта по разделителю
string obj_name[];
StringSplit(sparam, '_', obj_name);
//--- проверка, является ли объект кнопкой
if(obj_name[0]=="Button")
{
//--- получаем индекс кнопки
}
}
}

```

```

int index=(int)StringToInteger(obj_name[1]);
//--- установим кнопку в ненажатое состояние
ExtButtons[index].State(false);
//--- установим новое значения свойства в зависимости от его типа
if(ExtDataTypes[index]=='I')
    ChangeIntegerProperty(index);
if(ExtDataTypes[index]=='D')
    ChangeDoubleProperty(index);
if(ExtDataTypes[index]=='S')
    ChangeStringProperty(index);
}

}

//--- перерисовка значений свойств
RedrawProperties();
ChartRedraw();
}

//+-----+
//| Изменение целочисленного свойства графика |
//+-----+
void ChangeIntegerProperty(const int index)
{
//--- получаем текущее значение свойства
long value=ChartGetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index]);
//--- определяем следующее значение свойства
switch(ExtDrawTypes[index])
{
case 'C':
    value=GetNextColor((color)value);
    break;
default:
    value=(long)GetNextValue((double)value,index);
    break;
}
//--- устанавливаем новое значение свойства
ChartSetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index],0,value);
}

//+-----+
//| Изменение вещественного свойства графика |
//+-----+
void ChangeDoubleProperty(const int index)
{
//--- получаем текущее значение свойства
double value=ChartGetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index]);
//--- определяем следующее значение свойства
value=GetNextValue(value,index);
//--- устанавливаем новое значение свойства
ChartSetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index],value);
}

```

```

//| Изменение строкового свойства графика
//+-----+
void ChangeStringProperty(const int index)
{
//--- статическая переменная для переключения внутри массива комментариев ExtComments
    static uint comment_index=1;
//--- меняем индекс для получения другого комментария
    comment_index=1-comment_index;
//--- устанавливаем новое значение свойства
    ChartSetString(0, (ENUM_CHART_PROPERTY_STRING)ExtNumbers[index],ExtComments[comment_
    ]
//+-----+
//| Определение следующего значения свойства
//+-----+
double GetNextValue(const double value,const int index)
{
    if(value+ExtStep[index]<=Ext.MaxValue[index])
        return(value+ExtStep[index]);
    else
        return(Ext.MinValue[index]);
}
//+-----+
//| Получение следующего цвета для свойства типа color
//+-----+
color GetNextColor(const color clr)
{
//--- вернем следующее значение цвета
    switch(clr)
    {
        case clrWhite: return(clrRed);
        case clrRed:   return(clrGreen);
        case clrGreen: return(clrBlue);
        case clrBlue:  return(clrBlack);
        default:       return(clrWhite);
    }
}
//+-----+
//| Перерисовка значений свойств
//+-----+
void RedrawProperties(void)
{
//--- текст значения свойства
    string text;
    long   value;
//--- цикл по количеству свойств
    for(int i=0;i<ExtCount;i++)
    {
        text="";
        switch(ExtDataTypes[i])

```

```

{
    case 'I':
        //--- получаем текущее значение свойства
        if(!ChartGetInteger(0, (ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[i], 0, value))
            break;
        //--- текст целочисленного свойства
        switch(ExtDrawTypes[i])
        {
            //--- свойство цвета
            case 'C':
                text=(string)((color)value);
                break;
            //--- булевое свойство
            case 'B':
                text=(string)((bool)value);
                break;
            //--- свойство перечисления ENUM_CHART_MODE
            case 'M':
                text=EnumToString((ENUM_CHART_MODE)value);
                break;
            //--- свойство перечисления ENUM_CHART_VOLUME_MODE
            case 'V':
                text=EnumToString((ENUM_CHART_VOLUME_MODE)value);
                break;
            //--- число типа int
            default:
                text=IntegerToString(value);
                break;
        }
        break;
    case 'D':
        //--- текст вещественного свойства
        text=DoubleToString(ChartGetDouble(0, (ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[i]));
        break;
    case 'S':
        //--- текст строкового свойства
        text=ChartGetString(0, (ENUM_CHART_PROPERTY_STRING)ExtNumbers[i]);
        break;
    }
    //--- отобразим значение свойства
    ExtLabelsValue[i].Description(text);
}

//+-----+
//| Создание панели для управления свойствами графика |
//+-----+
bool PrepareControls(void)
{
    //--- выделим память под массивы с запасом
}

```

```

MemoryAllocation(LAST_PROPERTY_NUMBER+1);

//--- переменные
int i=0;           // переменная цикла
int col_1=0;        // количество свойств в первом столбце
int col_2=0;        // количество свойств во втором столбце
int col_3=0;        // количество свойств в третьем столбце
//--- текущее количество свойств - 0
ExtCount=0;
//--- ищем свойства в цикле
while(i<=LAST_PROPERTY_NUMBER)
{
    //--- запомним текущий номер свойства
    ExtNumbers[ExtCount]=i;
    //--- увеличим значение переменной цикла
    i++;
    //--- проверим, есть ли свойство с таким номером
    if(CheckNumber(ExtNumbers[ExtCount],ExtNames[ExtCount],ExtDataTypes[ExtCount],ExtCount))
    {
        //--- создаем элементы управления для свойства
        switch(ExtGroupTypes[ExtCount])
        {
            case 1:
                //--- создаем надписи и кнопку для свойства
                if(!ShowProperty(ExtCount,0,X_PROPERTY_NAME_1,X_PROPERTY_VALUE_1,X_BUTTON))
                    return(false);
                //--- количество элементов в первом столбце увеличилось
                col_1++;
                break;
            case 2:
                //--- создаем надписи и кнопку для свойства
                if(!ShowProperty(ExtCount,1,X_PROPERTY_NAME_2,X_PROPERTY_VALUE_2,X_BUTTON))
                    return(false);
                //--- количество элементов во втором столбце увеличилось
                col_2++;
                break;
            case 3:
                //--- создаем только надписи для свойства
                if(!ShowProperty(ExtCount,2,X_PROPERTY_NAME_2,X_PROPERTY_VALUE_2,0,Y_BUTTON))
                    return(false);
                //--- количество элементов в третьем столбце увеличилось
                col_3++;
                break;
        }
        //--- определим максимальное, минимальное значение свойства и шаг
        GetMaxMinStep(ExtNumbers[ExtCount],Ext.MaxValue[ExtCount],Ext.MinValue[ExtCount]);
        //--- увеличим количество свойств
        ExtCount++;
    }
}

```

```

//--- освободим память, которая не используется массивами
MemoryAllocation(ExtCount);

//--- перерисуем значения свойств
RedrawProperties();
ChartRedraw();

//--- успешное выполнение
return(true);
}

//+-----+
// | Выделение памяти для массивов |
//+-----+

void MemoryAllocation(const int size)
{
    ArrayResize(ExtLabelsName,size);
    ArrayResize(ExtLabelsValue,size);
    ArrayResize(ExtButtons,size);
    ArrayResize(ExtNumbers,size);
    ArrayResize(ExtNames,size);
    ArrayResize(ExtDataTypes,size);
    ArrayResize(ExtGroupTypes,size);
    ArrayResize(ExtDrawTypes,size);
    ArrayResize(Ext.MaxValue,size);
    ArrayResize(Ext.MinValue,size);
    ArrayResize(Ext.Step,size);
}

//+-----+
// | Проверяем индекс свойства на принадлежность к одному из |
// | перечислений ENUM_CHART_PROPERTIES |
//+-----+

bool CheckNumber(const int ind,string &name,uchar &data_type,uint &group_type,uchar &c
{
    //--- проверяем, является ли свойство целочисленным
    ResetLastError();
    name=EnumToString((ENUM_CHART_PROPERTY_INTEGER)ind);
    if(_LastError==0)
    {
        data_type='I';                                // свойство из перечисления ENUM_CHART_PROP
        GetTypes(ind,group_type,draw_type); // определим параметры отображения свойства
        return(true);
    }

    //--- проверяем, является ли свойство вещественным
    ResetLastError();
    name=EnumToString((ENUM_CHART_PROPERTY_DOUBLE)ind);
    if(_LastError==0)
    {
        data_type='D';                                // свойство из перечисления ENUM_CHART_PROP
        GetTypes(ind,group_type,draw_type); // определим параметры отображения свойства
        return(true);
    }
}

```

```

//--- проверяем, является ли свойство строковым
ResetLastError();
name=EnumToString((ENUM_CHART_PROPERTY_STRING)ind);
if(_LastError==0)
{
    data_type='S'; // свойство из перечисления ENUM_CHART_PROPERTY_STRING
    GetTypes(ind,group_type,draw_type); // определим параметры отображения свойства
    return(true);
}
//--- свойство не принадлежит ни одному перечислению
return(false);
}

//+-----+
//| Определение того, в какой группе должно находиться свойство, | |
//| и его тип отображения | |
//+-----+
void GetTypes(const int property_number,uint &group_type,uchar &draw_type)
{
//--- проверим на принадлежность свойства к третьей группе
//--- свойства третьей группы отображаются во втором столбце, начиная с CHART_BRING_TO_FRONT
if(CheckThirdGroup(property_number,group_type,draw_type))
    return;
//--- проверим на принадлежность свойства ко второй группе
//--- свойства второй группы отображаются во втором столбце с начала
if(CheckSecondGroup(property_number,group_type,draw_type))
    return;
//--- если попали сюда, значит свойство принадлежит первой группе (первый столбец)
CheckFirstGroup(property_number,group_type,draw_type);
}

//+-----+
//| Функция проверяет, принадлежит ли свойство третьей группе, и | |
//| если да, то определяет его тип отображения | |
//+-----+
bool CheckThirdGroup(const int property_number,uint &group_type,uchar &draw_type)
{
//--- проверим свойство на принадлежность к третьей группе
switch(property_number)
{
//--- булевые свойства
case CHART_IS_OBJECT:
case CHART_WINDOW_IS_VISIBLE:
    draw_type='B';
    break;
//--- целочисленные свойства
case CHART_VISIBLE_BARS:
case CHART_WINDOWS_TOTAL:
case CHART_WINDOW_HANDLE:
case CHART_WINDOW_YDISTANCE:
case CHART_FIRST_VISIBLE_BAR:
}
}

```

```

        case CHART_WIDTH_IN_BARS:
        case CHART_WIDTH_IN_PIXELS:
            draw_type='I';
            break;
            //--- вещественные свойства
        case CHART_PRICE_MIN:
        case CHART_PRICE_MAX:
            draw_type='D';
            break;
            //--- по сути, это свойство является командой показа графика поверх всех других
            //--- для данной панели в его применении нет необходимости, так как окно всегда
            //--- будет становиться поверх всех других раньше, чем мы используем его
        case CHART_BRING_TO_TOP:
            draw_type=' ';
            break;
            //--- свойство не принадлежит к третьей группе
        default:
            return(false);
        }
        //--- свойство принадлежит к третьей группе
        group_type=3;
        return(true);
    }

//+-----+
//| Функция проверяет, принадлежит ли свойство второй группе, и если |
//| да, то определяет его тип отображения |
//+-----+
bool CheckSecondGroup(const int property_number,uint &group_type,uchar &draw_type)
{
    //--- проверим свойство на принадлежность ко второй группе
    switch(property_number)
    {
        //--- свойство типа ENUM_CHART_MODE
        case CHART_MODE:
            draw_type='M';
            break;
            //--- свойство типа ENUM_CHART_VOLUME_MODE
        case CHART_SHOW_VOLUMES:
            draw_type='V';
            break;
            //--- строковое свойство
        case CHART_COMMENT:
            draw_type='S';
            break;
            //--- свойство цвета
        case CHART_COLOR_BACKGROUND:
        case CHART_COLOR_FOREGROUND:
        case CHART_COLOR_GRID:
        case CHART_COLOR_VOLUME:

```

```

        case CHART_COLOR_CHART_UP:
        case CHART_COLOR_CHART_DOWN:
        case CHART_COLOR_CHART_LINE:
        case CHART_COLOR_CANDLE_BULL:
        case CHART_COLOR_CANDLE_BEAR:
        case CHART_COLOR_BID:
        case CHART_COLOR_ASK:
        case CHART_COLOR_LAST:
        case CHART_COLOR_STOP_LEVEL:
            draw_type='C';
            break;
        //--- свойство не принадлежит ко второй группе
        default:
            return(false);
    }
//--- свойство принадлежит ко второй группе
group_type=2;
return(true);
}
//-----------------------------------------------------------------------------------------------------------------
//| Эта функция вызывается только в том случае, если уже известно, |
//| что свойство не принадлежит второй и третьей группам свойств |
//-----------------------------------------------------------------------------------------------------------------
void CheckFirstGroup(const int property_number,uint &group_type,uchar &draw_type)
{
//--- свойство принадлежит первой группе
group_type=1;
//--- определим тип отображения свойства
switch(property_number)
{
//--- целочисленные свойства
case CHART_SCALE:
case CHART_HEIGHT_IN_PIXELS:
    draw_type='I';
    return;
//--- вещественные свойства
case CHART_SHIFT_SIZE:
case CHART_FIXED_POSITION:
case CHART_FIXED_MAX:
case CHART_FIXED_MIN:
case CHART_POINTS_PER_BAR:
    draw_type='D';
    return;
//--- остались только булевые свойства
default:
    draw_type='B';
    return;
}
}

```

```

//+-----+
//| Создание надписи и кнопки для свойства |
//+-----+
bool ShowProperty(const int ind,const int type,const int x1,const int x2,
                  const int xb,const int y,const bool btn)
{
//--- статический массив для переключения внутри массива цвета ExtColors
    static uint color_index[3]={1,1,1};
//--- меняем индекс для получения другого цвета
    color_index[type]=1-color_index[type];
//--- выведем надписи и кнопку (если btn=true) для свойства
    if(!LabelCreate(ExtLabelsName[ind],"name_"+(string)ind,ExtNames[ind],ExtColors[color_index[0]],0,x1,y))
        return(false);
    if(!LabelCreate(ExtLabelsValue[ind],"value_"+(string)ind,"",ExtColors[color_index[1]],0,x2,y))
        return(false);
    if(btn && !ButtonCreate(ExtButtons[ind],(string)ind,xb,y+1))
        return(false);
//--- успешное выполнение
    return(true);
}

//+-----+
//| Создание надписи |
//+-----+
bool LabelCreate(CChartObjectLabel &lbl,const string name,const string text,
                 const color clr,const int x,const int y)
{
    if(!lbl.Create(0,"Label_"+name,0,x,y))
        return(false);
    if(!lbl.Description(text))
        return(false);
    if(!lbl.FontSize(10))
        return(false);
    if(!lbl.Color(clr))
        return(false);
//--- успешное выполнение
    return(true);
}

//+-----+
//| Создание кнопки |
//+-----+
bool ButtonCreate(CChartObjectButton &btn,const string name,
                  const int x,const int y)
{
    if(!btn.Create(0,"Button_"+name,0,x,y,50,15))
        return(false);
    if(!btn.Description("Next"))
        return(false);
    if(!btn.FontSize(10))
        return(false);
}

```

```
if(!btn.Color(clrBlack))
    return(false);
if(!btn.BackColor(clrWhite))
    return(false);
if(!btn.BorderColor(clrBlack))
    return(false);
//--- успешное выполнение
return(true);
}

//+-----+
//| Устанавливаем максимальное, минимальное значение свойства и шаг |
//+-----+
void GetMaxMinStep(const int property_number,double &max,double &min,double &step)
{
    double value;
//--- установим значения в зависимости от типа свойства
switch(property_number)
{
    case CHART_SCALE:
        max=5;
        min=0;
        step=1;
        break;
    case CHART_MODE:
    case CHART_SHOW_VOLUMES:
        max=2;
        min=0;
        step=1;
        break;
    case CHART_SHIFT_SIZE:
        max=50;
        min=10;
        step=2.5;
        break;
    case CHART_FIXED_POSITION:
        max=90;
        min=0;
        step=15;
        break;
    case CHART_POINTS_PER_BAR:
        max=19;
        min=1;
        step=3;
        break;
    case CHART_FIXED_MAX:
        value=ChartGetDouble(0,CHART_FIXED_MAX);
        max=value*1.25;
        min=value;
        step=value/32;
}
```

```
        break;
    case CHART_FIXED_MIN:
        value=ChartGetDouble(0,CHART_FIXED_MIN);
        max=value;
        min=value*0.75;
        step=value/32;
        break;
    case CHART_HEIGHT_IN_PIXELS:
        max=700;
        min=520;
        step=30;
        break;
        //--- значения по умолчанию
    default:
        max=1;
        min=0;
        step=1;
    }
}
```

## Константы объектов

Предусмотрено 44 графических объекта, которые можно создавать и отображать на ценовом графике. Все константы для работы с объектами разбиты на 9 групп:

- [Типы объектов](#) - идентификаторы графических объектов;
- [Свойства объектов](#) - работа со свойствами графических объектов;
- [Способы привязки объектов](#) - константы позиционирования объектов на графике;
- [Угол привязки](#) - позволяет установить угла графика, относительно которого производится позиционирование объекта в пикселях;
- [Видимость объектов](#) - задание таймфреймов, на которых объект является видимым;
- [Уровни волн Эллиотта](#) - градация волновой разметки;
- [Объекты Ганна](#) - константы тренда для веера Ганна и сетки Ганна;
- [Набор Web-цветов](#) - константы предопределенных Web-цветов;
- [Wingdings](#) - коды символов шрифта Wingdings.

## Типы объектов

При создании графического объекта функцией [ObjectCreate\(\)](#) необходимо указать тип создаваемого объекта, который может принимать одно из значений перечисления ENUM\_OBJECT. Дальнейшие уточнения [свойств](#) созданного объекта возможно с помощью функций по работе с [графическими объектами](#).

### ENUM\_OBJECT

Идентификатор		Описание
<a href="#">OBJ_VLINE</a>		Вертикальная линия
<a href="#">OBJ_HLINE</a>	—	Горизонтальная линия
<a href="#">OBJ_TREND</a>	/	Трендовая линия
<a href="#">OBJ_TRENDBYANGLE</a>	↖	Трендовая линия по углу
<a href="#">OBJ_CYCLES</a>		Циклические линии
<a href="#">OBJ_ARROWED_LINE</a>	↗	Объект "Линия со стрелкой"
<a href="#">OBJ_CHANNEL</a>	↔	Равноудаленный канал
<a href="#">OBJ_STDDEVCHANNEL</a>	↔	Канал стандартного отклонения
<a href="#">OBJ_REGRESSION</a>	↗↗	Канал на линейной регрессии
<a href="#">OBJ_PITCHFORK</a>		Вилы Эндрюса
<a href="#">OBJ_GANNLINE</a>	/G	Линия Ганна
<a href="#">OBJ_GANNFAN</a>	↙G	Веер Ганна
<a href="#">OBJ_GANNGRID</a>	☒G	Сетка Ганна
<a href="#">OBJ_FIBO</a>	☒F	Уровни Фибоначчи
<a href="#">OBJ_FIBOTIMES</a>	☒☒F	Временные зоны Фибоначчи
<a href="#">OBJ_FIBOFAN</a>	↗F	Веер Фибоначчи
<a href="#">OBJ_FIBOARC</a>	↙F	Дуги Фибоначчи
<a href="#">OBJ_FIBOCHANNEL</a>	↔F	Канал Фибоначчи
<a href="#">OBJ_EXPANSION</a>	☒☒F	Расширение Фибоначчи
<a href="#">OBJ_ELLIOTWAVE5</a>	↖E	5-волновка Эллиота
<a href="#">OBJ_ELLIOTWAVE3</a>	↗F	3-волновка Эллиота
<a href="#">OBJ_RECTANGLE</a>	█	Прямоугольник
<a href="#">OBJ_TRIANGLE</a>	▲	Треугольник
<a href="#">OBJ_ELLIPSE</a>	●	Эллипс

<u>OBJ_ARROW_THUMB_UP</u>		Знак "Хорошо" (большой палец вверх)
<u>OBJ_ARROW_THUMB_DOWN</u>		Знак "Плохо" (большой палец вниз)
<u>OBJ_ARROW_UP</u>		Знак "Стрелка вверх"
<u>OBJ_ARROW_DOWN</u>		Знак "Стрелка вниз"
<u>OBJ_ARROW_STOP</u>		Знак "Стоп"
<u>OBJ_ARROW_CHECK</u>		Знак "Птичка" (галка)
<u>OBJ_ARROW_LEFT_PRICE</u>		Левая ценовая метка
<u>OBJ_ARROW_RIGHT_PRICE</u>		Правая ценовая метка
<u>OBJ_ARROW_BUY</u>		Знак "Buy"
<u>OBJ_ARROW_SELL</u>		Знак "Sell"
<u>OBJ_ARROW</u>		Объект "Стрелка"
<u>OBJ_TEXT</u>		Объект "Текст"
<u>OBJ_LABEL</u>		Объект "Текстовая метка"
<u>OBJ_BUTTON</u>		Объект "Кнопка"
<u>OBJ_CHART</u>		Объект "График"
<u>OBJ_BITMAP</u>		Объект "Рисунок"
<u>OBJ_BITMAP_LABEL</u>		Объект "Графическая метка"
<u>OBJ_EDIT</u>		Объект "Поле ввода"
<u>OBJ_EVENT</u>		Объект "Событие", соответствующий событию в экономическом календаре
<u>OBJ_RECTANGLE_LABEL</u>		Объект "Прямоугольная метка" для создания и оформления пользовательского графического интерфейса.

## OBJ\_VLINE

Вертикальная линия.



### Примечание

При создании вертикальной линии, можно указать режим отображения линии на все окна графика (свойство [OBJPROP\\_RAY](#)).

### Пример

Следующий скрипт создает и перемещает на графике вертикальную линию. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Вертикальная линия\"."
#property description "Дата точки привязки задается в процентах от ширины"
#property description "окна графика в барах."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="VLine";           // Имя линии
input int         InpDate=25;                // Дата линии в %
input color       InpColor=clrRed;           // Цвет линии
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линии
input int         InpWidth=3;                // Толщина линии
input bool        InpBack=false;             // Линия на заднем плане
input bool        InpSelection=true;          // Выделить для перемещений
input bool        InpRay=true;               // Продолжение линии вниз

```

```

input bool           InpHidden=true;          // Скрыт в списке объектов
input long          InpZOrder=0;             // Приоритет на нажатие мышью
//+-----+
//| Создает вертикальную линию
//+-----+
bool VLineCreate(const long           chart_ID=0,           // ID графика
                  const string        name="VLine",         // имя линии
                  const int            sub_window=0,       // номер подокна
                  datetime            time=0,             // время линии
                  const color          clr=clrRed,        // цвет линии
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                  const int            width=1,            // толщина линии
                  const bool           back=false,         // на заднем плане
                  const bool           selection=true,    // выделить для перемещений
                  const bool           ray=true,          // продолжение линии вниз
                  const bool           hidden=true,        // скрыт в списке объектов
                  const long           z_order=0)         // приоритет на нажатие мыши
{
//--- если время линии не задано, то проводим ее через последний бар
  if(!time)
    time=TimeCurrent();
//---бросим значение ошибки
  ResetLastError();
//--- создадим вертикальную линию
  if(!ObjectCreate(chart_ID,name,OBJ_VLINE,sub_window,time,0))
  {
    Print(__FUNCTION__,
          ": не удалось создать вертикальную линию! Код ошибки = ",GetLastError());
    return(false);
  }
//--- установим цвет линии
  ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линии
  ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
  ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
  ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линии мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
  ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
  ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим отображения линии в подокнах графика
  ObjectSetInteger(chart_ID,name,OBJPROP_RAY,ray);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
  ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещение вертикальной линии |
//+-----+
bool VLineMove(const long    chart_ID=0,      // ID графика
               const string name="VLine", // имя линии
               datetime     time=0)      // время линии
{
//--- если время линии не задано, то перемещаем ее на последний бар
if(!time)
    time=TimeCurrent();
//--- сбросим значение ошибки
ResetLastError();
//--- переместим вертикальную линию
if(!ObjectMove(chart_ID,name,0,time,0))
{
    Print(__FUNCTION__,
          ": не удалось переместить вертикальную линию! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет вертикальную линию |
//+-----+
bool VLineDelete(const long    chart_ID=0,      // ID графика
                 const string name="VLine") // имя линии
{
//--- сбросим значение ошибки
ResetLastError();
//--- удалим вертикальную линию
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": не удалось удалить вертикальную линию! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
}

```

```

//--- проверим входные параметры на корректность
if(InpDate<0 || InpDate>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}

//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

//--- массив для хранения значений дат, которые будут использованы
//--- для установки и изменения координаты точки привязки линии
datetime date[];

//--- выделение памяти
ArrayResize(date,bars);

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- определим точки для рисования линии
int d=InpDate*(bars-1)/100;

//--- создадим вертикальную линию
if(!VLineCreate(0,InpName,0,date[d],InpColor,InpStyle,InpWidth,InpBack,
    InpSelection,InpRay,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать линию
//--- счетчик цикла
int h_steps=bars/2;
//--- перемещаем линию
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d<bars-1)
        d+=1;
    //--- сдвигаем точку
    if(!VLineMove(0,InpName,date[d]))
    {
        return;
    }

    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
    {
        return;
    }

    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.03 секунды
    Sleep(30);
}

```

```
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим канал с графика
VLineDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
}
```

## OBJ\_HLINE

Горизонтальная линия.



### Пример

Следующий скрипт создает и перемещает на графике горизонтальную линию. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Горизонтальная линия\"."
#property description "Цена точки привязки задается в процентах от высоты"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="HLine";           // Имя линии
input int         InpPrice=25;                // Цена линии в %
input color       InpColor=clrRed;            // Цвет линии
input ENUM_LINE_STYLE InpStyle=STYLE_DASH;    // Стиль линии
input int         InpWidth=3;                 // Толщина линии
input bool        InpBack=false;              // Линия на заднем плане
input bool        InpSelection=true;           // Выделить для перемещений
input bool        InpHidden=true;              // Скрыт в списке объектов
input long        InpZOrder=0;                // Приоритет на нажатие мышью
//+-----+
//| Создает горизонтальную линию |
//+-----+
```

```

bool HLineCreate(const long          chart_ID=0,           // ID графика
                 const string        name="HLine",          // имя линии
                 const int           sub_window=0,        // номер подокна
                 double              price=0,            // цена линии
                 const color         clr=clrRed,         // цвет линии
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                 const int           width=1,            // толщина линии
                 const bool          back=false,         // на заднем плане
                 const bool          selection=true,     // выделить для перемещений
                 const bool          hidden=true,        // скрыт в списке объектов
                 const long          z_order=0)          // приоритет на нажатие мыши

{
//--- если цена не задана, то установим ее на уровне текущей цены Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим горизонтальную линию
    if(!ObjectCreate(chart_ID,name,OBJ_HLINE,sub_window,0,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать горизонтальную линию! Код ошибки = ",GetLastError())
        return(false);
    }
//--- установим цвет линии
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линии мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
    return(true);
}

//-----+
//| Перемещение горизонтальной линии |
//-----+
bool HLineMove(const long   chart_ID=0,    // ID графика

```

```

        const string name="HLine", // имя линии
        double      price=0       // цена линии
    {
//--- если цена линии не задана, то перемещаем ее на уровень текущей цены Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
    ResetLastError();
//--- переместим горизонтальную линию
    if(!ObjectMove(chart_ID,name,0,0,price))
    {
        Print(__FUNCTION__,
              ": не удалось переместить горизонтальную линию! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Удаляет горизонтальную линию |
//+-----+
bool HLineDelete(const long   chart_ID=0,    // ID графика
                 const string name="HLine") // имя линии
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим горизонтальную линию
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить горизонтальную линию! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
    if(InpPrice<0 || InpPrice>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }
//--- размер массива price
    int accuracy=1000;

```

```

//--- массив для хранения значений цен, которые будут использованы
//--- для установки и изменения координаты точки привязки линии
    double price[];
//--- выделение памяти
    ArrayResize(price,accuracy);
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
//--- определим точки для рисования линии
    int p=InpPrice*(accuracy-1)/100;
//--- создадим горизонтальную линию
    if(!HLineCreate(0,InpName,0,price[p],InpColor,InpStyle,InpWidth,InpBack,
        InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
//--- перерисуем график и подождем 1 секунду
    ChartRedraw();
    Sleep(1000);
//--- теперь будем перемещать линию
//--- счетчик цикла
    int v_steps=accuracy/2;
//--- перемещаем линию
    for(int i=0;i<v_steps;i++)
    {
        //--- возьмем следующее значение
        if(p<accuracy-1)
            p+=1;
        //--- сдвигаем точку
        if(!HLineMove(0,InpName,price[p]))
            return;
        //--- проверим факт принудительного завершения скрипта
        if(IsStopped())
            return;
        //--- перерисуем график
        ChartRedraw();
    }
//--- задержка в 1 секунду
    Sleep(1000);
//--- удалим с графика
    HLineDelete(0,InpName);
    ChartRedraw();
//--- задержка в 1 секунду
    Sleep(1000);

```

```
//---
```

## OBJ\_TREND

Трендовая линия.



### Примечание

Для трендовой линии можно указать режим продолжения ее отображения вправо и/или влево (свойства [OBJPROP\\_RAY\\_RIGHT](#) и [OBJPROP\\_RAY\\_LEFT](#) соответственно).

### Пример

Следующий скрипт создает и перемещает на графике трендовую линию. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Трендовая линия\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="Trend";           // Имя линии
input int         InpDate1=35;                // Дата 1-ой точки в %
input int         InpPrice1=60;                // Цена 1-ой точки в %
input int         InpDate2=65;                // Дата 2-ой точки в %
input int         InpPrice2=40;                // Цена 2-ой точки в %
input color        InpColor=clrRed;            // Цвет линии
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линии

```

```

input int          InpWidth=2;           // Толщина линии
input bool         InpBack=false;        // Линия на заднем плане
input bool         InpSelection=true;    // Выделить для перемещений
input bool         InpRayLeft=false;     // Продолжение линии влево
input bool         InpRayRight=false;    // Продолжение линии вправо
input bool         InpHidden=true;       // Скрыт в списке объектов
input long         InpZOrder=0;          // Приоритет на нажатие мышью
//+-----+
//| Создает линию тренда по заданным координатам | 
//+-----+
bool TrendCreate(const long          chart_ID=0,          // ID графика
                  const string        name="TrendLine", // имя линии
                  const int           sub_window=0,      // номер подокна
                  datetime           time1=0,          // время первой точки
                  double              price1=0,         // цена первой точки
                  datetime           time2=0,          // время второй точки
                  double              price2=0,         // цена второй точки
                  const color         clr=clrRed,       // цвет линии
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                  const int           width=1,          // толщина линии
                  const bool          back=false,        // на заднем плане
                  const bool          selection=true,   // выделить для перемещений
                  const bool          ray_left=false,    // продолжение линии влево
                  const bool          ray_right=false,   // продолжение линии вправо
                  const bool          hidden=true,       // скрыт в списке объектов
                  const long          z_order=0)        // приоритет на нажатие мыши
{
//--- установим координаты точек привязки, если они не заданы
    ChangeTrendEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим трендовую линию по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_TREND,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": не удалось создать линию тренда! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет линии
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линии мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection

```

```

//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- включим (true) или отключим (false) режим продолжения отображения линии влево
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения линии вправо
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки линии тренда |
//+-----+
bool TrendPointChange(const long    chart_ID=0,           // ID графика
                      const string name="TrendLine", // имя линии
                      const int     point_index=0,   // номер точки привязки
                      datetime    time=0,          // координата времени точки привязки
                      double       price=0)        // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);

//---бросим значение ошибки
ResetLastError();

//--- переместим точку привязки линии тренда
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Функция удаляет линию тренда с графика. |
//+-----+
bool TrendDelete(const long    chart_ID=0,           // ID графика
                 const string name="TrendLine") // имя линии
{
//---бросим значение ошибки
ResetLastError();

//---удалим линию тренда
}

```

```

if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": не удалось удалить линию тренда! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Проверяет значения точек привязки линии тренда и для пустых |
//| значений устанавливает значения по умолчанию |
//+-----+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                            datetime &time2,double &price2)
{
//--- если время первой точки не задано, то она будет на текущем баре
if(!time1)
    time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
if(!price1)
    price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее второй
if(!time2)
{
//--- массив для приема времени открытия 10 последних баров
datetime temp[10];
CopyTime(Symbol(),Period(),time1,10,temp);
//--- установим вторую точку на 9 баров левее первой
time2=temp[0];
}
//--- если цена второй точки не задана, то она совпадает с ценой первой точки
if(!price2)
    price2=price1;
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
}

```

```

//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки линии
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования линии
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
//--- создадим линию тренда
if(!TrendCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,InpStyle,
    InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки линии
//--- счетчик цикла
int v_steps=accuracy/5;
//--- перемещаем первую точку привязки по вертикали
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1>1)
        p1-=1;
    //--- сдвигаем точку
    if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))

```

```

        return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
}

//--- перемещаем вторую точку привязки по вертикали
for(int i=0;i<v_steps;i++)
{
//--- возьмем следующее значение
if(p2<accuracy-1)
    p2+=1;
//--- сдвигаем точку
if(!TrendPointChange(0,InpName,1,date[d2],price[p2]))
    return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
}

//--- задержка в полсекунды
Sleep(500);
//--- счетчик цикла
int h_steps=bars/2;
//--- перемещаем обе точки привязки по горизонтали одновременно
for(int i=0;i<h_steps;i++)
{
//--- возьмем следующие значения
if(d1<bars-1)
    d1+=1;
if(d2>1)
    d2-=1;
//--- сдвигаем точки
if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))
    return;
if(!TrendPointChange(0,InpName,1,date[d2],price[p2]))
    return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.03 секунды
Sleep(30);
}

//--- задержка в 1 секунду
Sleep(1000);

```

```
//--- удалим трендовую линию  
TrendDelete(0, InpName);  
ChartRedraw();  
//--- задержка в 1 секунду  
Sleep(1000);  
//---  
}
```

## OBJ\_TREND\_BY\_ANGLE

Трендовая линия по углу.



### Примечание

Для трендовой линии можно указать режим продолжения ее отображения вправо и/или влево (свойства [OBJPROP\\_RAY\\_RIGHT](#) и [OBJPROP\\_RAY\\_LEFT](#) соответственно).

Для установки наклона линии можно использовать как угол, так и координаты второй точки привязки.

### Пример

Следующий скрипт создает и перемещает на графике трендовую линию. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Трендовая линия по углу\"."
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="Trend";           // Имя линии
input int         InpDate1=50;                // Дата 1-ой точки в %
input int         InpPrice1=75;               // Цена 1-ой точки в %
input int         InpAngle=0;                 // Угол наклона линии
input color       InpColor=clrRed;            // Цвет линии

```

```

input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линии
input int InpWidth=2; // Толщина линии
input bool InpBack=false; // Линия на заднем плане
input bool InpSelection=true; // Выделить для перемещений
input bool InpRayLeft=false; // Продолжение линии влево
input bool InpRayRight=true; // Продолжение линии вправо
input bool InpHidden=true; // Скрыт в списке объектов
input long InpZOrder=0; // Приоритет на нажатие мышью
//+-----+
//| Создает линию тренда по углу |
//+-----+
bool TrendByAngleCreate(const long chart_ID=0, // ID графика
                        const string name="TrendLine", // имя линии
                        const int sub_window=0, // номер подокна
                        datetime time=0, // время точки
                        double price=0, // цена точки
                        const double angle=45.0, // угол наклона
                        const color clr=clrRed, // цвет линии
                        const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                        const int width=1, // толщина линии
                        const bool back=false, // на заднем плане
                        const bool selection=true, // выделить для перемещения
                        const bool ray_left=false, // продолжение линии влево
                        const bool ray_right=true, // продолжение линии вправо
                        const bool hidden=true, // скрыт в списке объектов
                        const long z_order=0) // приоритет на нажатие мышью
{
//--- для того, чтобы было удобно перемещать трендовую линию мышью, создадим вторую точку
    datetime time2=0;
    double price2=0;
//--- установим координаты точек привязки, если они не заданы
    ChangeTrendEmptyPoints(time,price,time2,price2);
//--- сбросим значение ошибки
    ResetLastError();
//--- строим трендовую линию по 2-ум точкам
    if(!ObjectCreate(chart_ID,name,OBJ_TREND_BY_ANGLE,sub_window,time,price,time2,price2))
    {
        Print(__FUNCTION__,
              ": не удалось создать линию тренда! Код ошибки = ",GetLastError());
        return(false);
    }
//--- изменяем угол наклона трендовой линии; в процессе изменения угла, координата вто
//--- рой точки линии переопределится автоматически в соответствии с новым значением угла
    ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- установим цвет линии
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
}

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линии мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения линии влево
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения линии вправо
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет координаты точки привязки линии тренда |
//+-----+
bool TrendPointChange(const long chart_ID=0,           // ID графика
                      const string name="TrendLine", // имя линии
                      datetime time=0,             // координата времени точки привязки
                      double price=0)            // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
ResetLastError();
//--- переместим точку привязки линии тренда
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет угол наклона линии тренда |
//+-----+
bool TrendAngleChange(const long chart_ID=0,           // ID графика
                      const string name="TrendLine", // имя линии
                      double angle=0)             // угол наклона
{
    Print(__FUNCTION__,
          ": не удалось изменить угол наклона линии тренда! Код ошибки = ",GetLastError());
    return(false);
}

```

```

        const string name="TrendLine", // имя линии тренда
        const double angle=45)           // угол наклона линии тренда
    {
//--- сбросим значение ошибки
    ResetLastError();
//--- изменим угол наклона линии тренда
    if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
    {
        Print(__FUNCTION__,
            ": не удалось изменить угол наклона линии! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Удаляет линию тренда
//+-----+
bool TrendDelete(const long    chart_ID=0,           // ID графика
                 const string name="TrendLine") // имя линии
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим линию тренда
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
            ": не удалось удалить линию тренда! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Проверяет значения точек привязки линии тренда и для пустых
//| значений устанавливает значения по умолчанию
//+-----+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                            datetime &time2,double &price2)
{
//--- если время первой точки не задано, то она будет на текущем баре
    if(!time1)
        time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- установим координаты второй, вспомогательной точки
//--- вторая точка будет лежать левее на 9 баров и иметь ту же цену
    datetime second_point_time[10];
}

```

```

CopyTime(Symbol(),Period(),time1,10,second_point_time);
time2=second_point_time[0];
price2=price1;
}
//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100)
{
Print("Ошибка! Некорректные значения входных параметров!");
return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки линии
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
return;
}
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
price[i]=min_price+i*step;
//--- определим точки для рисования линии
int d1=InpDate1*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
//--- создадим линию тренда
if(!TrendByAngleCreate(0,InpName,0,date[d1],price[p1],InpAngle,InpColor,InpStyle,
InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
{
return;
}

```

```
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать и вращать линию
//--- счетчик цикла
int v_steps=accuracy/2;
//--- перемещаем точку привязки и изменяем угол наклона линии
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1>1)
        p1-=1;
    //--- сдвигаем точку
    if(!TrendPointChange(0,IcpName,date[d1],price[p1]))
        return;
    if(!TrendAngleChange(0,IcpName,18*(i+1)))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);
//--- удалим с графика
TrendDelete(0,IcpName);
ChartRedraw();

//--- задержка в 1 секунду
Sleep(1000);
//---
}
```

## OBJ\_CYCLES

Циклические линии.



### Примечание

Расстояние между линиями задается координатами времени двух точек привязки объекта.

### Пример

Следующий скрипт создает и перемещает на графике циклические линии. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит циклические линии на графике."
#property description "Координаты точек привязки задаются в процентах"
#property description "от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="Cycles";    // Имя объекта
input int         InpDate1=10;        // Дата 1-ой точки в %
input int         InpPrice1=45;       // Цена 1-ой точки в %
input int         InpDate2=20;        // Дата 2-ой точки в %
input int         InpPrice2=55;       // Цена 2-ой точки в %
input color        InpColor=clrRed;    // Цвет циклических линий
input ENUM_LINE_STYLE InpStyle=STYLE_DOT; // Стиль циклических линий
input int         InpWidth=1;         // Толщина циклических линий

```

```

input bool           InpBack=false;          // Объект на заднем плане
input bool           InpSelection=true;       // Выделить для перемещений
input bool           InpHidden=true;          // Скрыт в списке объектов
input long           InpZOrder=0;            // Приоритет на нажатие мышью
//+-----+
//| Создает циклические линии
//+-----+
bool CyclesCreate(const long           chart_ID=0,          // ID графика
                   const string        name="Cycles",      // имя объекта
                   const int            sub_window=0,       // номер подокна
                   const datetime       time1=0,          // время первой точки
                   double               price1=0,          // цена первой точки
                   const datetime       time2=0,          // время второй точки
                   double               price2=0,          // цена второй точки
                   const color          clr=clrRed,        // цвет циклических линий
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль циклических линий
                   const int             width=1,           // толщина циклических линий
                   const bool            back=false,         // на заднем плане
                   const bool            selection=true,     // выделить для перемещений
                   const bool            hidden=true,        // скрыт в списке объектов
                   const long            z_order=0)        // приоритет на нажатие мыши
{
//--- установим координаты точек привязки, если они не заданы
    ChangeCyclesEmptyPoints(time1,price1,time2,price2);
//---бросим значение ошибки
    ResetLastError();
//--- создадим циклические линии по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_CYCLES,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": не удалось создать циклические линии! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет линий
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линий
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линий мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

```

```

//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID, name, OBJPROP_ZORDER, z_order);
//--- успешное выполнение
return(true);
}

//+-----+
// | Перемещает точку привязки
//+-----+
bool CyclesPointChange(const long chart_ID=0, // ID графика
                       const string name="Cycles", // имя объекта
                       const int point_index=0, // номер точки привязки
                       datetime time=0, // координата времени точки привязки
                       double price=0) // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(), SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID, name, point_index, time, price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ", GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
// | Удаляет циклические линии
//+-----+
bool CyclesDelete(const long chart_ID=0, // ID графика
                  const string name="Cycles") // имя объекта
{
//---бросим значение ошибки
ResetLastError();
//---удалим циклические линии
if(!ObjectDelete(chart_ID, name))
{
    Print(__FUNCTION__,
          ": не удалось удалить циклические линии! Код ошибки = ", GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

```

//+-----+
//| Проверяет значения точек привязки циклических линий и для пустых |
//| значений устанавливает значения по умолчанию |
//+-----+
void ChangeCyclesEmptyPoints(datetime &time1,double &price1,
                             datetime &time2,double &price2)
{
//--- если время первой точки не задано, то она будет на текущем баре
    if(!time1)
        time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее второй
    if(!time2)
    {
//--- массив для приема времени открытия 10 последних баров
        datetime temp[10];
        CopyTime(Symbol(),Period(),time1,10,temp);
//--- установим вторую точку на 9 баров левее первой
        time2=temp[0];
    }
//--- если цена второй точки не задана, то она совпадает с ценой первой точки
    if(!price2)
        price2=price1;
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
       InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }
//--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
    int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки циклических линий
    datetime date[];
    double price[];
//--- выделение памяти
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
}

```

```

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования циклических линий
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;

//--- создадим линию тренда
if(!CyclesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

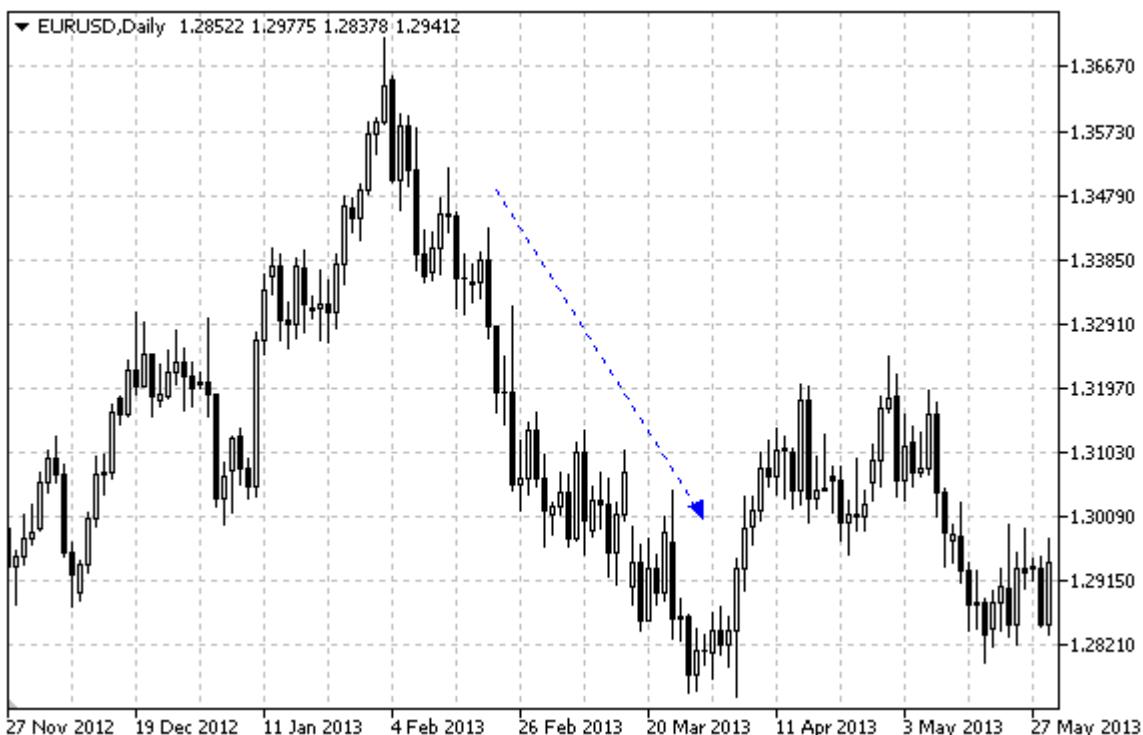
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
int h_steps=bars/5;
//--- перемещаем вторую точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d2<bars-1)
        d2+=1;
    //--- сдвигаем точку
    if(!CyclesPointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}

```

```
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
h_steps=bars/4;
//--- перемещаем первую точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d1<bars-1)
        d1+=1;
    //--- сдвигаем точку
    if(!CyclesPointChange(0,InpName,0,date[d1],price[p1]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим объект с графика
CyclesDelete(0,InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_ARROWED\_LINE

Линия со стрелкой.



### Пример

Следующий скрипт создает и перемещает на графике линию со стрелкой. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Линия со стрелкой\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="ArrowedLine"; // Имя линии
input int         InpDate1=35;           // Дата 1-ой точки в %
input int         InpPrice1=60;          // Цена 1-ой точки в %
input int         InpDate2=65;          // Дата 2-ой точки в %
input int         InpPrice2=40;          // Цена 2-ой точки в %
input color       InpColor=clrRed;        // Цвет линии
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линии
input int         InpWidth=2;            // Толщина линии
input bool        InpBack=false;         // Линия на заднем плане
input bool        InpSelection=true;       // Выделить для перемещений
input bool        InpHidden=true;         // Скрыт в списке объектов
input long        InpZOrder=0;           // Приоритет на нажатие мышью
```

```

//+-----+
//| Создает линию со стрелкой по заданным координатам |
//+-----+
bool ArrowedLineCreate(const long           chart_ID=0,          // ID графика
                       const string        name="ArrowedLine", // имя линии
                       const int            sub_window=0,      // номер подокна
                       const datetime       time1=0,          // время первой точки
                       const double         price1=0,         // цена первой точки
                       const datetime       time2=0,          // время второй точки
                       const double         price2=0,         // цена второй точки
                       const color          clr=clrRed,       // цвет линии
                       const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                       const int             width=1,          // толщина линии
                       const bool            back=false,        // на заднем плане
                       const bool            selection=true,    // выделить для перемещения
                       const bool            hidden=true,       // скрыт в списке объектов
                       const long            z_order=0)        // приоритет на нажатие

{
//--- установим координаты точек привязки, если они не заданы
    ChangeArrowedLineEmptyPoints(time1,price1,time2,price2);
//---бросим значение ошибки
    ResetLastError();
//--- создадим линию со стрелкой по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_ARROWED_LINE,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": не удалось создать линию со стрелкой! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет линии
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линии мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
    return(true);
}

```

```

}

//+-----+
//| Перемещает точку привязки линии со стрелкой |
//+-----+

bool ArrowedLinePointChange(const long    chart_ID=0,           // ID графика
                            const string name="ArrowedLine", // имя линии
                            const int     point_index=0,      // номер точки привязки
                            datetime    time=0,             // координата времени точки
                            double       price=0)          // координата цены точки

{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
    ResetLastError();
//--- переместим точку привязки линии
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

//+-----+
//| Функция удаляет линию со стрелкой с графика |
//+-----+

bool ArrowedLineDelete(const long    chart_ID=0,           // ID графика
                      const string name="ArrowedLine") // имя линии

{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим линию со стрелкой
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить линию со стрелкой! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

//+-----+
//| Проверяет значения точек привязки линии и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+

```

```

void ChangeArrowedLineEmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2)
{
//--- если время первой точки не задано, то она будет на текущем баре
if(!time1)
    time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
if(!price1)
    price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее первой
if(!time2)
{
//--- массив для приема времени открытия 10 последних баров
datetime temp[10];
CopyTime(Symbol(),Period(),time1,10,temp);
//--- установим вторую точку на 9 баров левее первой
time2=temp[0];
}
//--- если цена второй точки не задана, то она совпадает с ценой первой точки
if(!price2)
    price2=price1;
}

//-----+
//| Script program start function |
//-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки линии
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
}

```

```

Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
return;
}
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования линии
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
//--- создадим линию со стрелкой
if(!ArrowedLineCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
    InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки линии
//--- счетчик цикла
int v_steps=accuracy/5;
//--- перемещаем вторую точку привязки по вертикали
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p2<accuracy-1)
        p2+=1;
    //--- сдвигаем точку
    if(!ArrowedLinePointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- перемещаем первую точку привязки по вертикали
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1>1)
        p1-=1;
}

```

```
//--- сдвигаем точку
if(!ArrowedLinePointChange(0,InpName,0,date[d1],price[p1]))
    return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
}

//--- задержка в полсекунды
Sleep(500);
//--- счетчик цикла
int h_steps=bars/2;
//--- перемещаем обе точки привязки по горизонтали одновременно
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующие значения
    if(d1<bars-1)
        d1+=1;
    if(d2>1)
        d2-=1;
    //--- сдвигаем точки
    if(!ArrowedLinePointChange(0,InpName,0,date[d1],price[p1]))
        return;
    if(!ArrowedLinePointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.03 секунды
    Sleep(30);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим линию со стрелкой
ArrowedLineDelete(0,InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_CHANNEL

Равноудаленный канал.



### Примечание

Для равноудаленного канала можно указать режим продолжения его отображения вправо и/или влево (свойства [OBJPROP\\_RAY\\_RIGHT](#) и [OBJPROP\\_RAY\\_LEFT](#) соответственно). Также можно установить режим заливки канала цветом.

### Пример

Следующий скрипт создает и перемещает на графике равноудаленный канал. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Равноудаленный канал\"."
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="Channel";    // Имя канала
input int         InpDate1=25;          // Дата 1-ой точки в %
input int         InpPrice1=60;         // Цена 1-ой точки в %
input int         InpDate2=65;          // Дата 2-ой точки в %
input int         InpPrice2=80;         // Цена 2-ой точки в %
input int         InpDate3=30;          // Дата 3-ей точки в %

```

```

input int          InpPrice3=40;           // Цена 3-ей точки в %
input color        InpColor=clrRed;        // Цвет канала
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линий канала
input int          InpWidth=2;            // Толщина линий канала
input bool         InpBack=false;         // Канал на заднем плане
input bool         InpFill=false;          // Заливка канала цветом
input bool         InpSelection=true;       // Выделить для перемещений
input bool         InpRayLeft=false;        // Продолжение канала влево
input bool         InpRayRight=false;       // Продолжение канала вправо
input bool         InpHidden=true;          // Скрыт в списке объектов
input long         InpZOrder=0;            // Приоритет на нажатие мышью
//+-----+
//| Создает равноудаленный канал по заданным координатам |+
//+-----+
bool ChannelCreate(const long          chart_ID=0,           // ID графика
                    const string        name="Channel",      // имя канала
                    const int           sub_window=0,        // номер подокна
                    const datetime       time1=0,           // время первой точки
                    double              price1=0,          // цена первой точки
                    const datetime       time2=0,           // время второй точки
                    double              price2=0,          // цена второй точки
                    const datetime       time3=0,           // время третьей точки
                    double              price3=0,          // цена третьей точки
                    const color          clr=clrRed,         // цвет канала
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий канала
                    const int           width=1,            // толщина линий канала
                    const bool          fill=false,         // заливка канала цветом
                    const bool          back=false,         // на заднем плане
                    const bool          selection=true,     // выделить для перемещений
                    const bool          ray_left=false,      // продолжение канала влево
                    const bool          ray_right=false,    // продолжение канала вправо
                    const bool          hidden=true,        // скрыт в списке объектов
                    const long          z_order=0)          // приоритет на нажатие мышью
{
//--- установим координаты точек привязки, если они не заданы
    ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//---бросим значение ошибки
    ResetLastError();
//--- создадим канал по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_CHANNEL,sub_window,time1,price1,time2,price2,time3,price3))
    {
        Print(__FUNCTION__,
              ": не удалось создать равноудаленный канал! Код ошибки = ",GetLastError())
        return(false);
    }
//--- установим цвет канала
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий канала
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);

```

```

//--- установим толщину линий канала
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки канала
ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения канала для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения канала влево
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения канала вправо
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
// | Перемещает точку привязки канала
//+-----+
bool ChannelPointChange(const long chart_ID=0,           // ID графика
                        const string name="Channel", // имя канала
                        const int    point_index=0,   // номер точки привязки
                        datetime   time=0,          // координата времени точки привязки
                        double     price=0)         // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

```

//+-----+
//| Удаляет канал
//+-----+
bool ChannelDelete(const long    chart_ID=0,          // ID графика
                    const string name="Channel") // имя канала
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим канал
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить канал! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Проверяет значения точек привязки канала и для пустых значений |
//| устанавливает значения по умолчанию                                |
//+-----+
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                               double &price2,datetime &time3,double &price3)
{
//--- если время второй (правой) точки не задано, то она будет на текущем баре
    if(!time2)
        time2=TimeCurrent();
//--- если цена второй точки не задана, то она будет иметь значение Bid
    if(!price2)
        price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой (левой) точки не задано, то она лежит на 9 баров левее второй
    if(!time1)
    {
//--- массив для приема времени открытия 10 последних баров
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
//--- установим первую точку на 9 баров левее второй
        time1=temp[0];
    }
//--- если цена первой точки не задана, то сдвинем ее на 300 пунктов выше второй
    if(!price1)
        price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время третьей точки не задано, то оно совпадает с временем первой точки
    if(!time3)
        time3=time1;
//--- если цена третьей точки не задана, то она совпадает с ценой второй точки
    if(!price3)
        price3=price2;
}

```

```

    }

//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    //--- проверим входные параметры на корректность
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
       InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
       InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }

    //--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
    //--- размер массива price
    int accuracy=1000;
    //--- массивы для хранения значений дат и цен, которые будут использованы
    //--- для установки и изменения координат точек привязки канала
    datetime date[];
    double price[];
    //--- выделение памяти
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);
    //--- заполним массив дат
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
        return;
    }

    //--- заполним массив цен
    //--- найдем максимальное и минимальное значение графика
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
    //--- определим шаг изменения цены и заполним массив
    double step=(max_price-min_price)/accuracy;
    for(int i=0;i<accuracy;i++)
        price[i]=min_price+i*step;
    //--- определим точки для рисования канала
    int d1=InpDate1*(bars-1)/100;
    int d2=InpDate2*(bars-1)/100;
    int d3=InpDate3*(bars-1)/100;
    int p1=InpPrice1*(accuracy-1)/100;
    int p2=InpPrice2*(accuracy-1)/100;
    int p3=InpPrice3*(accuracy-1)/100;
    //--- создадим равноудаленный канал
    if(!ChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price

```

```

InpStyle, InpWidth, InpFill, InpBack, InpSelection, InpRayLeft, InpRayRight, InpHidden,
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки канала
//--- счетчик цикла
int h_steps=bars/6;
//--- перемещаем вторую точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d2<bars-1)
        d2+=1;
    //--- сдвигаем точку
    if(!ChannelPointChange(0, InpName, 1, date[d2], price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- перемещаем первую точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d1>1)
        d1-=1;
    //--- сдвигаем точку
    if(!ChannelPointChange(0, InpName, 0, date[d1], price[p1]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла

```

```
int v_steps=accuracy/10;
//--- перемещаем третью точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p3>1)
        p3-=1;
    //--- сдвигаем точку
    if(!ChannelPointChange(0,IInpName,2,date[d3],price[p3]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим канал с графика
ChannelDelete(0,IInpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_STDDEVCHANNEL

Канал стандартного отклонения.



### Примечание

Для канала стандартного отклонения можно указать режим продолжения его отображения вправо и/или влево (свойства [OBJPROP\\_RAY\\_RIGHT](#) и [OBJPROP\\_RAY\\_LEFT](#) соответственно). Также можно установить режим заливки канала цветом.

Для изменения значения отклонения канала используется свойство [OBJPROP\\_DEVIATION](#).

### Пример

Следующий скрипт создает и перемещает на графике канал стандартного отклонения. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Канал стандартного отклонения\""
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="StdDevChannel";    // Имя канала
input int         InpDate1=10;                  // Дата 1-ой точки в %
input int         InpDate2=40;                  // Дата 2-ой точки в %
input double      InpDeviation=1.0;            // Отклонение
input color       InpColor=clrRed;              // Цвет канала

```

## Константы, перечисления и структуры

```

input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий канала
input int InpWidth=2; // Толщина линий канала
input bool InpFill=false; // Заливка канала цветом
input bool InpBack=false; // Канал на заднем плане
input bool InpSelection=true; // Выделить для перемещений
input bool InpRayLeft=false; // Продолжение канала влево
input bool InpRayRight=false; // Продолжение канала вправо
input bool InpHidden=true; // Скрыт в списке объектов
input long InpZOrder=0; // Приоритет на нажатие мышью
//+-----+
//| Создает канал стандартного отклонения по заданным координатам |
//+-----+
bool StdDevChannelCreate(const long
                           const string
                           const int
                           datetime
                           datetime
                           const double
                           const color
                           const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий канала
                           const int
                           width=1,
                           const bool
                           fill=false,
                           const bool
                           back=false,
                           const bool
                           selection=true,
                           const bool
                           ray_left=false,
                           const bool
                           ray_right=false,
                           const bool
                           hidden=true,
                           const long
                           z_order=0) // ID графика
{
    chart_ID=0, // ID графика
    name="Channel", // имя канала
    sub_window=0, // номер подокна
    time1=0, // время первой точки
    time2=0, // время второй точки
    deviation=1.0, // отклонение
    clr=clrRed, // цвет канала
    chart_ID, // ID графика
    name, // имя канала
    sub_window, // номер подокна
    time1, // время первой точки
    time2, // время второй точки
    deviation, // отклонение
    clr, // цвет канала
    width, // толщина линий канала
    fill, // заливка канала цветом
    back, // на заднем плане
    selection, // выделить для перемещений
    ray_left, // продолжение канала влево
    ray_right, // продолжение канала вправо
    hidden, // скрыт в списке объектов
    z_order // приоритет на нажатие мышью
}

//--- установим координаты точек привязки, если они не заданы
ChangeChannelEmptyPoints(time1,time2);
//---бросим значение ошибки
ResetLastError();
//--- создадим канал по заданным координатам
if(!ObjectCreate(chart_ID,name,OBJ_STDDEVCHANNEL,sub_window,time1,0,time2,0))
{
    Print(__FUNCTION__,
          "не удалось создать канал стандартного отклонения! Код ошибки = ",GetLastError());
    return(false);
}
//--- установим величину отклонения, от нее зависит ширина канала
ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation);
//--- установим цвет канала
ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий канала
ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий канала
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки канала

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения канала для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения канала влево
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения канала вправо
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки канала |
//+-----+
bool StdDevChannelPointChange(const long chart_ID=0, // ID графика
                               const string name="Channel", // имя канала
                               const int point_index=0, // номер точки привязки
                               datetime time=0) // координата времени точки
{
//--- если время точки не задано, то перемещаем ее на текущий бар
if(!time)
    time=TimeCurrent();
//--- сбросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,0))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет отклонение канала |
//+-----+
bool StdDevChannelDeviationChange(const long chart_ID=0, // ID графика
                                   const string name="Channel", // имя канала
                                   const double deviation=1.0) // отклонение

```

```

{
//--- сбросим значение ошибки
ResetLastError();

//--- изменим угол наклона линии тренда
if(!ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation))
{
Print(__FUNCTION__,
": не удалось изменить отклонение канала! Код ошибки = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет канал |
//+-----+
bool StdDevChannelDelete(const long    chart_ID=0,          // ID графика
                         const string name="Channel") // имя канала
{
//--- сбросим значение ошибки
ResetLastError();

//--- удалим канал
if(!ObjDelete(chart_ID,name))
{
Print(__FUNCTION__,
": не удалось удалить канал! Код ошибки = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Проверяет значения точек привязки канала и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeChannelEmptyPoints(datetime &time1,datetime &time2)
{
//--- если время второй точки не задано, то она будет на текущем баре
if(!time2)
time2=TimeCurrent();

//--- если время первой точки не задано, то она лежит на 9 баров левее второй
if(!time1)
{
//--- массив для приема времени открытия 10 последних баров
datetime temp[10];
CopyTime(Symbol(),Period(),time2,10,temp);
//--- установим вторую точку на 9 баров левее второй
time1=temp[0];
}
}

```

```

    }

//+-----+
//| Script program start function | 
//+-----+

void OnStart()
{
    //--- проверим входные параметры на корректность
    if(InpDate1<0 || InpDate1>100 ||
       InpDate2<0 || InpDate2>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }

//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

//--- размер массива price
int accuracy=1000;

//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки канала
datetime date[];
double   price[];

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования канала
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;

//--- создадим канал стандартного отклонения
if(!StdDevChannelCreate(0,InpName,0,date[d1],date[d2],InpDeviation,InpColor,InpStyle,
InpWidth,InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду

```

```
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать канал по горизонтали вправо и расширять его
//--- счетчик цикла
int h_steps=bars/2;
//--- перемещаем канал
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующие значения
    if(d1<bars-1)
        d1+=1;
    if(d2<bars-1)
        d2+=1;
    //--- переместим точки привязки
    if(!StdDevChannelPointChange(0,IInpName,0,date[d1]))
        return;
    if(!StdDevChannelPointChange(0,IInpName,1,date[d2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
double v_steps=IInpDeviation*2;
//--- расширим канал
for(double i=IInpDeviation;i<v_steps;i+=10.0/accuracy)
{
    if(!StdDevChannelDeviationChange(0,IInpName,i))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим канал с графика
StdDevChannelDelete(0,IInpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

{}

## OBJ\_REGRESSION

Канал на линейной регрессии.



### Примечание

Для канала на линейной регрессии можно указать режим продолжения его отображения вправо и/или влево (свойства `OBJPROP_RAY_RIGHT` и `OBJPROP_RAY_LEFT` соответственно). Также можно установить режим заливки канала цветом.

### Пример

Следующий скрипт создает и перемещает на графике канал на линейной регрессии. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Канал на линейной регрессии\""
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="Regression"; // Имя канала
input int         InpDate1=10;           // Дата 1-ой точки в %
input int         InpDate2=40;           // Дата 2-ой точки в %
input color        InpColor=clrRed;       // Цвет канала
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линий канала
input int         InpWidth=2;            // Толщина линий канала

```

```

input bool           InpFill=false;          // Заливка канала цветом
input bool           InpBack=false;          // Канал на заднем плане
input bool           InpSelection=true;       // Выделить для перемещений
input bool           InpRayLeft=false;        // Продолжение канала влево
input bool           InpRayRight=false;       // Продолжение канала вправо
input bool           InpHidden=true;          // Скрыт в списке объектов
input long           InpZOrder=0;            // Приоритет на нажатие мышью
//+-----+
//| Создает канал на линейной регрессии по заданным координатам | |
//+-----+
bool RegressionCreate(const long           chart_ID=0,          // ID графика
                      const string         name="Regression", // имя канала
                      const int            sub_window=0,      // номер подокна
                      datetime            timel=0,          // время первой точки
                      datetime            time2=0,          // время второй точки
                      const color          clr=clrRed,        // цвет канала
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий канала
                      const int            width=1,          // толщина линий канала
                      const bool           fill=false,        // заливка канала цветом
                      const bool           back=false,       // на заднем плане
                      const bool           selection=true,    // выделить для перемещений
                      const bool           ray_left=false,   // продолжение канала влево
                      const bool           ray_right=false,  // продолжение канала вправо
                      const bool           hidden=true,      // скрыт в списке объектов
                      const long           z_order=0)        // приоритет на нажатие мышью
{
//--- установим координаты точек привязки, если они не заданы
    ChangeRegressionEmptyPoints(timel,time2);
//---бросим значение ошибки
    ResetLastError();
//--- создадим канал по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_REGRESSION,sub_window,timel,0,time2,0))
    {
        Print(__FUNCTION__,
              ": не удалось создать канал на линейной регрессии! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет канала
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий канала
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий канала
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки канала
    ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения канала для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект

```

```

//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения канала влево
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения канала вправо
    ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
    return(true);
}

//+-----+
//| Перемещает точку привязки канала |+
//+-----+
bool RegressionPointChange(const long chart_ID=0, // ID графика
                           const string name="Channel", // имя канала
                           const int point_index=0, // номер точки привязки
                           datetime time=0) // координата времени точки привязки
{
//--- если время точки не задано, то перемещаем ее на текущий бар
    if(!time)
        time=TimeCurrent();
//--- сбросим значение ошибки
    ResetLastError();
//--- переместим точку привязки
    if(!ObjectMove(chart_ID,name,point_index,time,0))
    {
        Print(__FUNCTION__,
              ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

//+-----+
//| Удаляет канал |+
//+-----+
bool RegressionDelete(const long chart_ID=0, // ID графика
                      const string name="Channel") // имя канала
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим канал
    if(!ObjectDelete(chart_ID,name))
    {

```

```

Print(__FUNCTION__,
      ": не удалось удалить канал! Код ошибки = ", GetLastError());
return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Проверяет значения точек привязки канала и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeRegressionEmptyPoints(datetime &time1,datetime &time2)
{
//--- если время второй точки не задано, то она будет на текущем баре
if(!time2)
  time2=TimeCurrent();
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
if(!time1)
{
  //--- массив для приема времени открытия 10 последних баров
  datetime temp[10];
  CopyTime(Symbol(),Period(),time2,10,temp);
  //--- установим первую точку на 9 баров левее второй
  time1=temp[0];
}
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 ||
   InpDate2<0 || InpDate2>100)
{
  Print("Ошибка! Некорректные значения входных параметров!");
  return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки канала
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
}

```

```

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования канала
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;

//--- создадим канал на линейной регрессии
if(!RegressionCreate(0,InpName,0,date[d1],date[d2],InpColor,InpStyle,InpWidth,
    InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать канал по горизонтали вправо
//--- счетчик цикла
int h_steps=bars/2;
//--- перемещаем канал
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующие значения
    if(d1<bars-1)
        d1+=1;
    if(d2<bars-1)
        d2+=1;

    //--- переместим точки привязки
    if(!RegressionPointChange(0,InpName,0,date[d1]))
        return;
    if(!RegressionPointChange(0,InpName,1,date[d2]))
        return;

    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;

    //--- перерисуем график
    ChartRedraw();

    // задержка в 0.05 секунды
}

```

```
    Sleep(50);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим канал с графика
RegressionDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
}
```

## OBJ\_PITCHFORK

Вилы Эндрюса.



### Примечание

Для "Вил Эндрюса" можно указать режим продолжения отображения вправо и/или влево (свойства [OBJPROP\\_RAY\\_RIGHT](#) и [OBJPROP\\_RAY\\_LEFT](#) соответственно).

Также можно указать количество линий-уровней, их значения и цвет.

### Пример

Следующий скрипт создает и перемещает на графике "Вилы Эндрюса". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Вилы Эндрюса\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="Pitchfork";    // Имя вил
input int         InpDate1=14;           // Дата 1-ой точки в %
input int         InpPrice1=40;          // Цена 1-ой точки в %
input int         InpDate2=18;           // Дата 2-ой точки в %
input int         InpPrice2=50;          // Цена 2-ой точки в %
input int         InpDate3=18;           // Дата 3-ей точки в %
input int         InpPrice3=30;          // Цена 3-ей точки в %

```

```

input color           InpColor=clrRed;          // Цвет вил
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID;    // Стиль линий вил
input int             InpWidth=1;              // Толщина линий вил
input bool            InpBack=false;           // Вилы на заднем плане
input bool            InpSelection=true;         // Выделить для перемещений
input bool            InpRayLeft=false;          // Продолжение вил влево
input bool            InpRayRight=false;         // Продолжение вил вправо
input bool            InpHidden=true;            // Скрыт в списке объектов
input long            InpZOrder=0;              // Приоритет на нажатие мышью
//+-----+
//| Создает "Вилы Эндрюса" по заданным координатам |
//+-----+
bool PitchforkCreate(const long           chart_ID=0,          // ID графика
                      const string        name="Pitchfork", // имя вил
                      const int           sub_window=0,       // номер подокна
                      datetime            time1=0,           // время первой точки
                      double              price1=0,          // цена первой точки
                      datetime            time2=0,           // время второй точки
                      double              price2=0,          // цена второй точки
                      datetime            time3=0,           // время третьей точки
                      double              price3=0,          // цена третьей точки
                      const color          clr=clrRed,         // цвет линий вил
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий вил
                      const int           width=1,           // толщина линий вил
                      const bool           back=false,          // на заднем плане
                      const bool           selection=true,        // выделить для перемещения
                      const bool           ray_left=false,        // продолжение вил влево
                      const bool           ray_right=false,       // продолжение вил вправо
                      const bool           hidden=true,          // скрыт в списке объектов
                      const long           z_order=0)          // приоритет на нажатие
{
//--- установим координаты точек привязки, если они не заданы
    ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим "Вилы Эндрюса" по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_PITCHFORK,sub_window,time1,price1,time2,price2,
                     time3,price3))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Вилы Эндрюса\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения вил для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения вил влево
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения вил вправо
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Задает количество уровней "Вил Эндрюса" и их параметры |+
//+-----+
bool PitchforkLevelsSet(int levels, // количество линий уровня
                        double values[], // значения линий уровня
                        color colors[], // цвет линий уровня
                        ENUM_LINE_STYLE &styles[], // стиль линий уровня
                        int widths[], // толщина линий уровня
                        const long chart_ID=0, // ID графика
                        const string name="Pitchfork") // имя вил
{
//--- проверим размеры массивов
if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
   levels!=ArraySize(widths) || levels!=ArraySize(widths))
{
Print(__FUNCTION__,"": длина массива не соответствуют количеству уровней, ошибка");
return(false);
}
//--- установим количество уровней
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- установим свойства уровней в цикле
for(int i=0;i<levels;i++)
{
//--- значение уровня
ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
//--- цвет уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
//--- стиль уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
//--- толщина уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
}
}

```

```

//--- описание уровня
ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1));
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки "Вил Эндрюса" |
//+-----+
bool PitchforkPointChange(const long    chart_ID=0,           // ID графика
                           const string name="Pitchfork", // имя канала
                           const int     point_index=0,   // номер точки привязки
                           datetime    time=0,          // координата времени точки
                           double       price=0)        // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет "Вилы Эндрюса" |
//+-----+
bool PitchforkDelete(const long    chart_ID=0,           // ID графика
                     const string name="Pitchfork") // имя канала
{
//---бросим значение ошибки
ResetLastError();
//--- удалим канал
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": не удалось удалить \"Вилы Эндрюса\"! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

```

    }

//+-----+
//| Проверяет значения точек привязки "Вил Эндрюса" и для пустых |
//| значений устанавливает значения по умолчанию |
//+-----+

void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                               double &price2,datetime &time3,double &price3)
{
    //--- если время второй (правой верхней) точки не задано, то она будет на текущем баре
    if(!time2)
        time2=TimeCurrent();

    //--- если цена второй точки не задана, то она будет иметь значение Bid
    if(!price2)
        price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);

    //--- если время первой (левой) точки не задано, то она лежит на 9 баров левее второй
    if(!time1)
    {
        //--- массив для приема времени открытия 10 последних баров
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- установим первую точку на 9 баров левее второй
        time1=temp[0];
    }

    //--- если цена первой точки не задана, то сдвинем ее на 200 пунктов ниже второй
    if(!price1)
        price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);

    //--- если время третьей точки не задано, то оно совпадает с временем второй точки
    if(!time3)
        time3=time2;

    //--- если цена третьей точки не задана, то сдвинем ее на 200 пунктов ниже первой
    if(!price3)
        price3=price1-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
    }

//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
    //--- проверим входные параметры на корректность
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
       InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
       InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }

    //--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
    //--- размер массива price
}

```

```

int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки "Вил Эндрюса"
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования "Вил Эндрюса"
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int d3=InpDate3*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
int p3=InpPrice3*(accuracy-1)/100;
//--- создадим вилы
if(!PitchforkCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price[p3],
    InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки вил
//--- счетчик цикла
int v_steps=accuracy/10;
//--- перемещаем первую точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1>1)
        p1-=1;
    //--- сдвигаем точку
}

```

```

        if(!PitchforkPointChange(0, InpName, 0, date[d1], price[p1]))
            return;
        //--- проверим факт принудительного завершения скрипта
        if(IsStopped())
            return;
        //--- перерисуем график
        ChartRedraw();
    }

    //--- задержка в 1 секунду
    Sleep(1000);
    //--- счетчик цикла
    int h_steps=bars/8;
    //--- перемещаем третью точку привязки
    for(int i=0;i<h_steps;i++)
    {
        //--- возьмем следующее значение
        if(d3<bars-1)
            d3+=1;
        //--- сдвигаем точку
        if(!PitchforkPointChange(0, InpName, 2, date[d3], price[p3]))
            return;
        //--- проверим факт принудительного завершения скрипта
        if(IsStopped())
            return;
        //--- перерисуем график
        ChartRedraw();
        //--- перерисуем график
        ChartRedraw();
        // задержка в 0.05 секунды
        Sleep(50);
    }
    //--- задержка в 1 секунду
    Sleep(1000);
    //--- счетчик цикла
    v_steps=accuracy/10;
    //--- перемещаем вторую точку привязки
    for(int i=0;i<v_steps;i++)
    {
        //--- возьмем следующее значение
        if(p2>1)
            p2-=1;
        //--- сдвигаем точку
        if(!PitchforkPointChange(0, InpName, 1, date[d2], price[p2]))
            return;
        //--- проверим факт принудительного завершения скрипта
        if(IsStopped())
            return;
        //--- перерисуем график
        ChartRedraw();
    }
}

```

```
}

//--- задержка в 1 секунду
Sleep(1000);

//--- удалим вилы с графика
PitchforkDelete(0, InpName);
ChartRedraw();

//--- задержка в 1 секунду
Sleep(1000);

//---

}
```

## OBJ\_GANNLINE

Линия Ганна.



### Примечание

Для "Линии Ганна" можно указать режим продолжения ее отображения вправо и/или влево (свойства `OBJPROP_RAY_RIGHT` и `OBJPROP_RAY_LEFT` соответственно).

Для установки наклона линии можно использовать как угол Ганна с масштабом, так и координаты второй точки привязки.

### Пример

Следующий скрипт создает и перемещает на графике "Линию Ганна". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Линия Ганна\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="GannLine";           // Имя линии
input int         InpDate1=20;                  // Дата 1-ой точки в %
input int         InpPrice1=75;                 // Цена 1-ой точки в %
input int         InpDate2=80;                  // Дата 2-ой точки в %
input double      InpAngle=0.0;                // Угол Ганна

```

```

input double          InpScale=1.0;           // Масштаб
input color           InpColor=clrRed;         // Цвет линии
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линии
input int              InpWidth=2;            // Толщина линии
input bool             InpBack=false;          // Линия на заднем плане
input bool             InpSelection=true;        // Выделить для перемещений
input bool             InpRayLeft=false;        // Продолжение линии влево
input bool             InpRayRight=true;         // Продолжение линии вправо
input bool             InpHidden=true;          // Скрыт в списке объектов
input long             InpZOrder=0;            // Приоритет на нажатие мышью
//+-----+
//| Создает "Линию Ганна" по координатам, углу и масштабу |
//+-----+
bool GannLineCreate(const long          chart_ID=0,          // ID графика
                     const string       name="GannLine",      // имя линии
                     const int          sub_window=0,        // номер подокна
                     datetime          time1=0,           // время первой точки
                     double             price1=0,          // цена первой точки
                     datetime          time2=0,           // время второй точки
                     const double       angle=1.0,          // угол Ганна
                     const double       scale=1.0,          // масштаб
                     const color         clr=clrRed,        // цвет линии
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                     const int          width=1,           // толщина линии
                     const bool          back=false,         // на заднем плане
                     const bool          selection=true,       // выделить для перемещений
                     const bool          ray_left=false,      // продолжение линии влево
                     const bool          ray_right=true,       // продолжение линии вправо
                     const bool          hidden=true,         // скрыт в списке объектов
                     const long          z_order=0)          // приоритет на нажатие мышью
{
//--- установим координаты точек привязки, если они не заданы
    ChangeGannLineEmptyPoints(time1,price1,time2);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим "Линию Ганна" по заданным координатам
//--- правильная координата цены второй точки привязки переопределится
//--- автоматически после изменения угла Ганна и (или) масштаба,
    if(!ObjectCreate(chart_ID,name,OBJ_GANNLINE,sub_window,time1,price1,time2,0))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Линию Ганна\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- изменим угол Ганна
    ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- изменим масштаб (количество пиксов на бар)
    ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- установим цвет линии

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линии
ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения линии для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения линии влево
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения линии вправо
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки "Линии Ганна" |
//+-----+
bool GannLinePointChange(const long chart_ID=0, // ID графика
                        const string name="GannLine", // имя линии
                        const int point_index=0, // номер точки привязки
                        datetime time=0, // координата времени точки привязки
                        double price=0) // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки линии
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

```

}

//+-----+
//| Изменяет угол Ганна
//+-----+
bool GannLineAngleChange(const long    chart_ID=0,          // ID графика
                         const string name="GannLine", // имя линии
                         const double  angle=1.0)     // угол Ганна
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- изменим угол Ганна
    if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
    {
        Print(__FUNCTION__,
              ": не удалось изменить угол Ганна! Код ошибки = ",GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}
//+-----+
//| Изменяет масштаб "Линии Ганна"
//+-----+
bool GannLineScaleChange(const long    chart_ID=0,          // ID графика
                         const string name="GannLine", // имя линии
                         const double  scale=1.0)     // масштаб
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- изменим масштаб (количество пипсов на бар)
    if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
    {
        Print(__FUNCTION__,
              ": не удалось изменить масштаб! Код ошибки = ",GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}
//+-----+
//| Функция удаляет "Линию Ганна" с графика
//+-----+
bool GannLineDelete(const long    chart_ID=0,          // ID графика
                    const string name="GannLine") // имя линии
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- удалим линию Ганна
    if(!ObjectDelete(chart_ID,name))

```

```

{
    Print(__FUNCTION__,
        ": не удалось удалить \"Линию Ганна\"! Код ошибки = ", GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Проверяет значения точек привязки "Линии Ганна" и для пустых      |
//| значений устанавливает значения по умолчанию                         |
//+-----+
void ChangeGannLineEmptyPoints(datetime &time1,double &price1,datetime &time2)
{
//--- если время второй точки не задано, то она будет на текущем баре
if(!time2)
    time2=TimeCurrent();
//--- если время первой точки не задано, то она лежит на 9 баров левее первой
if(!time1)
{
//--- массив для приема времени открытия 10 последних баров
datetime temp[10];
CopyTime(Symbol(),Period(),time2,10,temp);
//--- установим первую точку на 9 баров левее второй
time1=temp[0];
}
//--- если цена первой точки не задана, то она будет иметь значение Bid
if(!price1)
    price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//+-----+
//| Script program start function                                         |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки линии
datetime date[];
}

```

```

        double    price[];
//--- выделение памяти
        ArrayResize(date,bars);
        ArrayResize(price,accuracy);
//--- заполним массив дат
        ResetLastError();
        if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
        {
            Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
            return;
        }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
        double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
        double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
        double step=(max_price-min_price)/accuracy;
        for(int i=0;i<accuracy;i++)
            price[i]=min_price+i*step;
//--- определим точки для рисования линии Ганна
        int d1=InpDate1*(bars-1)/100;
        int d2=InpDate2*(bars-1)/100;
        int p1=InpPrice1*(accuracy-1)/100;
//--- создадим линию Ганна
        if(!GannLineCreate(0,InpName,0,date[d1],price[p1],date[d2],InpAngle,InpScale,InpColor,
                           InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpOrder))
        {
            return;
        }
//--- перерисуем график и подождем 1 секунду
        ChartRedraw();
        Sleep(1000);
//--- теперь будем перемещать точку привязки линии и менять угол
//--- счетчик цикла
        int v_steps=accuracy/2;
//--- перемещаем первую точку привязки по вертикали
        for(int i=0;i<v_steps;i++)
        {
//--- возьмем следующее значение
            if(p1>1)
                p1-=1;
//--- сдвигаем точку
            if(!GannLinePointChange(0,InpName,0,date[d1],price[p1]))
                return;
//--- проверим факт принудительного завершения скрипта
            if(IsStopped())
                return;
//--- перерисуем график
            ChartRedraw();
        }
    }
}

```

```
}

//--- задержка в полсекунды
Sleep(500);

//--- определим текущее значение угла Ганна (изменился
//--- после перемещения первой точки привязки)
double curr_angle;
if(!ObjectGetDouble(0, InpName, OBJPROP_ANGLE, 0, curr_angle))
    return;

//--- счетчик цикла
v_steps=accuracy/8;
//--- изменяем угол Ганна
for(int i=0;i<v_steps;i++)
{
    if(!GannLineAngleChange(0, InpName, curr_angle-0.05*i))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);

//--- удалим линию с графика
GannLineDelete(0, InpName);
ChartRedraw();

//--- задержка в 1 секунду
Sleep(1000);

//---
```

## OBJ\_GANNFAN

Веер Ганна.



### Примечание

Для "Веера Ганна" можно указать тип тренда из перечисления [ENUM\\_GANN\\_DIRECTION](#). Регулируя значение масштаба ([OBJPROP\\_SCALE](#)), можно менять угол наклона линий веера.

### Пример

Следующий скрипт создает и перемещает на графике "Веер Ганна". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Веер Ганна\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="GannFan";           // Имя веера
input int              InpDate1=15;                // Дата 1-ой точки в %
input int              InpPrice1=25;                // Цена 1-ой точки в %
input int              InpDate2=85;                // Дата 2-ой точки в %
input double            InpScale=2.0;               // Масштаб
input bool              InpDirection=false;          // Направление тренда
input color             InpColor=clrRed;             // Цвет веера

```

```

input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий веера
input int InpWidth=1; // Толщина линий веера
input bool InpBack=false; // Веер на заднем плане
input bool InpSelection=true; // Выделить для перемещений
input bool InpHidden=true; // Скрыт в списке объектов
input long InpZOrder=0; // Приоритет на нажатие мышью
//+-----+
//| Создает "Веер Ганна"
//+-----+
bool GannFanCreate(const long chart_ID=0, // ID графика
                    const string name="GannFan", // имя веера
                    const int sub_window=0, // номер подокна
                    datetime time1=0, // время первой точки
                    double price1=0, // цена первой точки
                    datetime time2=0, // время второй точки
                    const double scale=1.0, // масштаб
                    const bool direction=true, // направление тренда
                    const color clr=clrRed, // цвет веера
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий веера
                    const int width=1, // толщина линий веера
                    const bool back=false, // на заднем плане
                    const bool selection=true, // выделить для перемещений
                    const bool hidden=true, // скрыт в списке объектов
                    const long z_order=0) // приоритет на нажатие мышью
{
//--- установим координаты точек привязки, если они не заданы
    ChangeGannFanEmptyPoints(time1,price1,time2);
//---бросим значение ошибки
    ResetLastError();
//--- создадим "Веер Ганна" по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_GANNFAN,sub_window,time1,price1,time2,0))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Веер Ганна\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- изменим масштаб (количество пипсов на бар)
    ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- изменим направление тренда "Веера Ганна" (true - нисходящий, false - восходящий)
    ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
//--- установим цвет веера
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линий веера
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий веера
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения веера для перемещений
}

```

```

//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки "Веера Ганна" |
//+-----+
bool GannFanPointChange(const long    chart_ID=0,      // ID графика
                        const string name="GannFan", // имя веера
                        const int     point_index=0, // номер точки привязки
                        datetime    time=0,        // координата времени точки привязки
                        double       price=0)      // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
ResetLastError();
//--- переместим точку привязки веера
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет масштаб "Веера Ганна" |
//+-----+
bool GannFanScaleChange(const long    chart_ID=0,      // ID графика
                        const string name="GannFan", // имя веера
                        const double scale=1.0)    // масштаб
{
//--- сбросим значение ошибки
ResetLastError();
//--- изменим масштаб (количество пипсов на бар)
if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))

```

```

{
    Print(__FUNCTION__,
          ": не удалось изменить масштаб! Код ошибки = ", GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
// | Изменяет направление тренда "Веера Ганна" |
//+-----+
bool GannFanDirectionChange(const long    chart_ID=0,      // ID графика
                            const string name="GannFan", // имя веера
                            const bool    direction=true) // направление тренда
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- изменим направление тренда "Веера Ганна"
    if(!ObjectSetInteger(chart_ID, name, OBJPROP_DIRECTION, direction))
    {
        Print(__FUNCTION__,
              ": не удалось изменить направление тренда! Код ошибки = ", GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

//+-----+
// | Функция удаляет "Веер Ганна" с графика |
//+-----+
bool GannFanDelete(const long    chart_ID=0,      // ID графика
                   const string name="GannFan") // имя веера
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- удалим веер Ганна
    if(!ObjectDelete(chart_ID, name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить \"Веер Ганна\"! Код ошибки = ", GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

//+-----+
// | Проверяет значения точек привязки "Веера Ганна" и для пустых |
// | значений устанавливает значения по умолчанию |
//+-----+

```

```

void ChangeGannFanEmptyPoints(datetime &time1,double &price1,datetime &time2)
{
//--- если время второй точки не задано, то она будет на текущем баре
if(!time2)
    time2=TimeCurrent();
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
if(!time1)
{
//--- массив для приема времени открытия 10 последних баров
datetime temp[10];
CopyTime(Symbol(),Period(),time2,10,temp);
//--- установим первую точку на 9 баров левее второй
time1=temp[0];
}
//--- если цена первой точки не задана, то она будет иметь значение Bid
if(!price1)
    price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//-----+
//| Script program start function |+
//-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100)
{
Print("Ошибка! Некорректные значения входных параметров!");
return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки веера
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
return;
}
//--- заполним массив цен
}

```

```

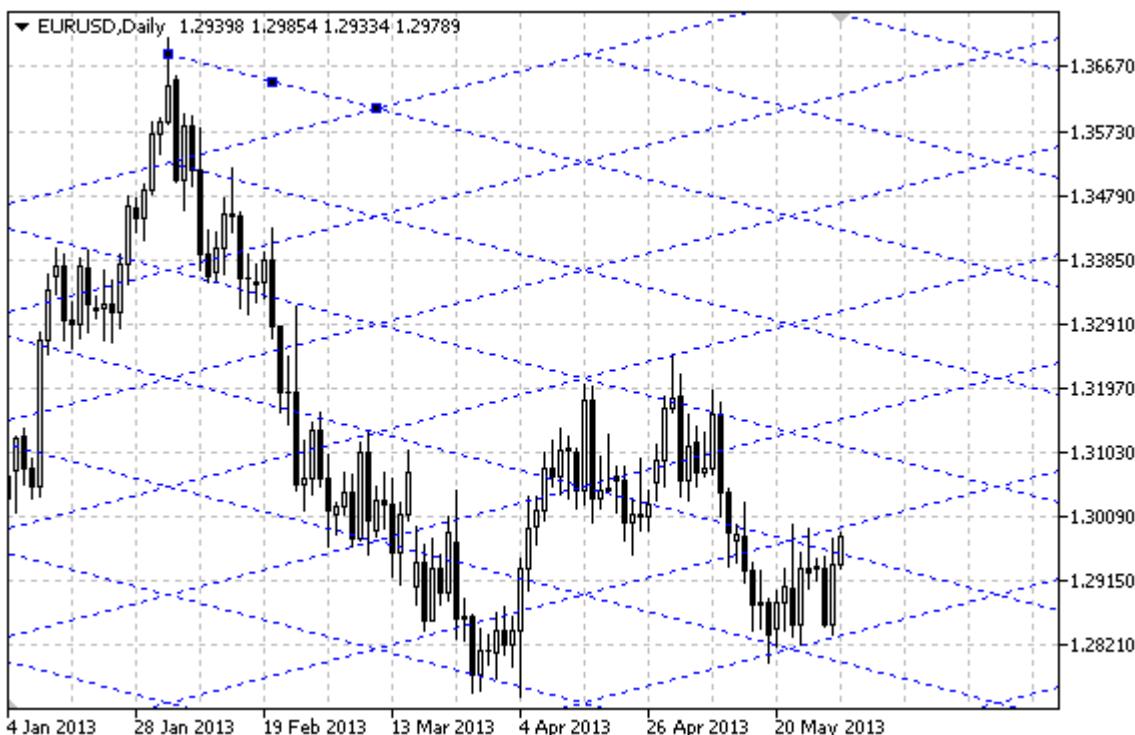
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования веера Ганна
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
//--- создадим веер Ганна
if(!GannFanCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpDirection,
    InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точку привязки веера
//--- счетчик цикла
int v_steps=accuracy/2;
//--- перемещаем первую точку привязки по вертикали
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1<accuracy-1)
        p1+=1;
    //--- сдвигаем точку
    if(!GannFanPointChange(0,InpName,0,date[d1],price[p1]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- изменим направление тренда веера на нисходящее
GannFanDirectionChange(0,InpName,true);
//--- перерисуем график
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим веер с графика
GannFanDelete(0,InpName);
ChartRedraw();

```

```
//--- задержка в 1 секунду
Sleep(1000);
//---
}
```

## OBJ\_GANNGRID

Сетка Ганна.



### Примечание

Для "Сетки Ганна" можно указать тип тренда из перечисления [ENUM\\_GANN\\_DIRECTION](#). Регулируя значение масштаба ([OBJPROP\\_SCALE](#)), можно менять угол наклона линий сетки.

### Пример

Следующий скрипт создает и перемещает на графике "Сетку Ганна". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Сетка Ганна\"."
#property description "Координаты точек привязки сетки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="GannGrid";           // Имя сетки
input int         InpDate1=15;                  // Дата 1-ой точки в %
input int         InpPrice1=25;                 // Цена 1-ой точки в %
input int         InpDate2=35;                  // Дата 2-ой точки в %
input double     InpScale=3.0;                  // Масштаб
input bool        InpDirection=false;          // Направление тренда
input color       InpColor=clrRed;              // Цвет сетки

```

```

input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий сетки
input int InpWidth=1; // Толщина линий веера
input bool InpBack=false; // Сетка на заднем плане
input bool InpSelection=true; // Выделить для перемещений
input bool InpHidden=true; // Скрыт в списке объектов
input long InpZOrder=0; // Приоритет на нажатие мышью
//+-----+
//| Создает "Сетку Ганна"
//+-----+
bool GannGridCreate(const long chart_ID=0, // ID графика
                     const string name="GannGrid", // имя сетки
                     const int sub_window=0, // номер подокна
                     datetime timel=0, // время первой точки
                     double pricel=0, // цена первой точки
                     datetime time2=0, // время второй точки
                     const double scale=1.0, // масштаб
                     const bool direction=true, // направление тренда
                     const color clr=clrRed, // цвет сетки
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий сетки
                     const int width=1, // толщина линий сетки
                     const bool back=false, // на заднем плане
                     const bool selection=true, // выделить для перемещений
                     const bool hidden=true, // скрыт в списке объектов
                     const long z_order=0) // приоритет на нажатие мышью
{
    //--- установим координаты точек привязки, если они не заданы
    ChangeGannGridEmptyPoints(timel,pricel,time2);
    //---бросим значение ошибки
    ResetLastError();
    //--- создадим "Сетку Ганна" по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_GANNGRID,sub_window,timel,pricel,time2,0))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Сетку Ганна\"! Код ошибки = ",GetLastError());
        return(false);
    }
    //--- изменим масштаб (количество пипсов на бар)
    ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
    //--- изменим направление тренда "Сетки Ганна" (true - нисходящий, false - восходящий)
    ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
    //--- установим цвет сетки
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- установим стиль отображения линий сетки
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- установим толщину линий сетки
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- включим (true) или отключим (false) режим выделения сетки для перемещений
}

```

```

//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки "Сетки Ганна" |
//+-----+
bool GannGridPointChange(const long    chart_ID=0,          // ID графика
                         const string name="GannGrid", // имя объекта "Сетка Ганна"
                         const int    point_index=0,   // номер точки привязки
                         datetime   time=0,         // координата времени точки привязки
                         double      price=0)       // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
ResetLastError();
//--- переместим точку привязки сетки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет масштаб "Сетки Ганна" |
//+-----+
bool GannGridScaleChange(const long    chart_ID=0,          // ID графика
                         const string name="GannGrid", // имя сетки
                         const double scale=1.0)     // масштаб
{
//--- сбросим значение ошибки
ResetLastError();
//--- изменим масштаб (количество пипсов на бар)
if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))

```

```

{
    Print(__FUNCTION__,
          ": не удалось изменить масштаб! Код ошибки = ", GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
// | Изменяет направление тренда "Сетки Ганна" |
//+-----+
bool GannGridDirectionChange(const long chart_ID=0,           // ID графика
                             const string name="GannGrid", // имя сетки
                             const bool   direction=true) // направление тренда
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- изменим направление тренда "Сетки Ганна"
    if(!ObjectSetInteger(chart_ID, name, OBJPROP_DIRECTION, direction))
    {
        Print(__FUNCTION__,
              ": не удалось изменить направление тренда! Код ошибки = ", GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

//+-----+
// | Функция удаляет "Сетку Ганна" с графика |
//+-----+
bool GannGridDelete(const long chart_ID=0,           // ID графика
                    const string name="GannGrid") // имя сетки
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- удалим сетку Ганна
    if(!ObjectDelete(chart_ID, name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить \"Сетку Ганна\"! Код ошибки = ", GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

//+-----+
// | Проверяет значения точек привязки "Сетки Ганна" и для пустых |
// | значений устанавливает значения по умолчанию |
//+-----+

```

```

void ChangeGannGridEmptyPoints(datetime &time1,double &price1,datetime &time2)
{
//--- если время второй точки не задано, то она будет на текущем баре
if(!time2)
    time2=TimeCurrent();
//--- если время первой точки не задано, то она лежит на 9 баров левее первой
if(!time1)
{
//--- массив для приема времени открытия 10 последних баров
datetime temp[10];
CopyTime(Symbol(),Period(),time2,10,temp);
//--- установим первую точку на 9 баров левее второй
time1=temp[0];
}
//--- если цена первой точки не задана, то она будет иметь значение Bid
if(!price1)
    price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//-----+
//| Script program start function |+
//-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100)
{
Print("Ошибка! Некорректные значения входных параметров!");
return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки сетки
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
return;
}
//--- заполним массив цен
}

```

```

//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования сетки Ганна
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
//--- создадим сетку Ганна
if(!GannGridCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpDirection,
    InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки сетки
//--- счетчик цикла
int v_steps=accuracy/4;
//--- перемещаем первую точку привязки по вертикали
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1<accuracy-1)
        p1+=1;
    if(!GannGridPointChange(0,InpName,0,date[d1],price[p1]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
int h_steps=bars/4;
//--- перемещаем вторую точку привязки по горизонтали
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d2<bars-1)
        d2+=1;
    if(!GannGridPointChange(0,InpName,1,date[d2],0))
        return;
}

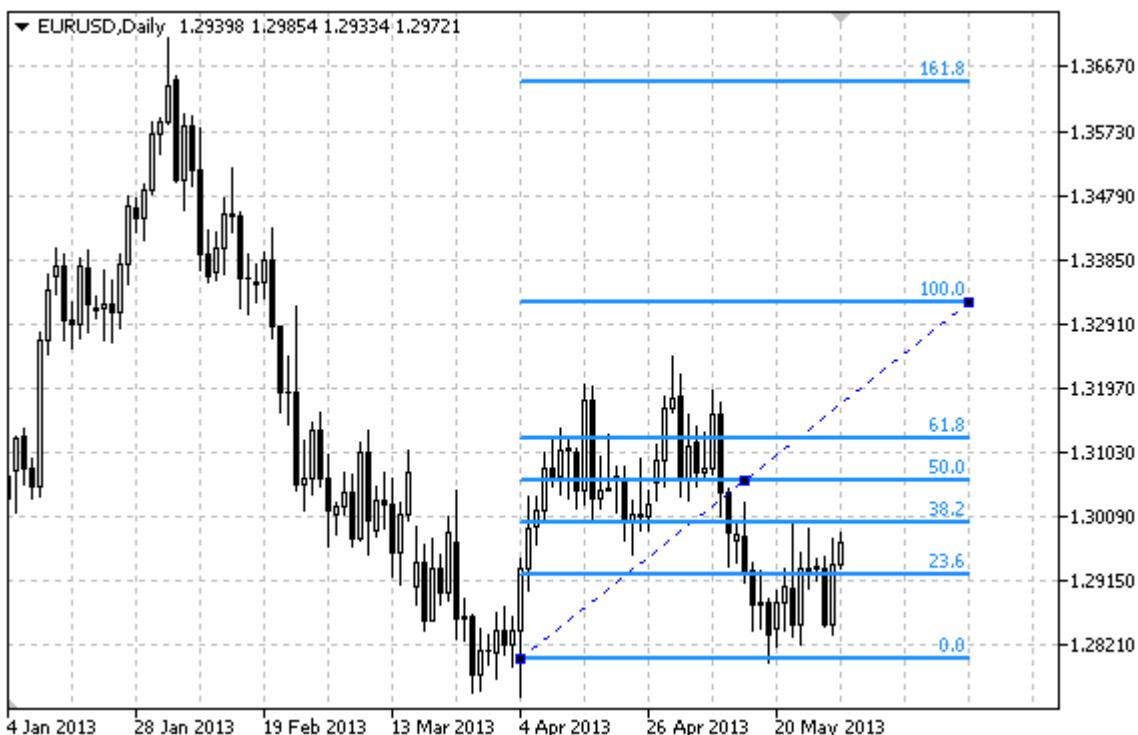
```

```
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}

//--- задержка в 1 секунду
Sleep(1000);
//--- изменим направление тренда сетки на нисходящее
GannGridDirectionChange(0, InpName, true);
//--- перерисуем график
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим сетку с графика
GannGridDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_FIBO

Уровни Фибоначчи.



### Примечание

Для "Уровней Фибоначчи" можно указать режим продолжения отображения вправо и/или влево (свойства `OBJPROP_RAY_RIGHT` и `OBJPROP_RAY_LEFT` соответственно).

Также можно указать количество линий-уровней, их значения и цвет.

### Пример

Следующий скрипт создает и перемещает на графике "Уровни Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Уровни Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="FiboLevels";           // Имя объекта
input int         InpDate1=10;                      // Дата 1-ой точки в %
input int         InpPrice1=65;                     // Цена 1-ой точки в %
input int         InpDate2=90;                     // Дата 2-ой точки в %
input int         InpPrice2=85;                     // Цена 2-ой точки в %
input color       InpColor=clrRed;                  // Цвет объекта
```

```

input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линии
input int InpWidth=2; // Толщина линии
input bool InpBack=false; // Объект на заднем плане
input bool InpSelection=true; // Выделить для перемещений
input bool InpRayLeft=false; // Продолжение объекта влево
input bool InpRayRight=false; // Продолжение объекта вправо
input bool InpHidden=true; // Скрыт в списке объектов
input long InpZOrder=0; // Приоритет на нажатие мышью
//+-----+
//| Создает "Уровни Фибоначчи" по заданным координатам |+
//+-----+
bool FiboLevelsCreate(const long chart_ID=0, // ID графика
                      const string name="FiboLevels", // имя объекта
                      const int sub_window=0, // номер подокна
                      datetime timel=0, // время первой точки
                      double price1=0, // цена первой точки
                      datetime time2=0, // время второй точки
                      double price2=0, // цена второй точки
                      const color clr=clrRed, // цвет объекта
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии объекта
                      const int width=1, // толщина линии объекта
                      const bool back=false, // на заднем плане
                      const bool selection=true, // выделить для перемещений
                      const bool ray_left=false, // продолжение объекта
                      const bool ray_right=false, // продолжение объекта
                      const bool hidden=true, // скрыт в списке объектов
                      const long z_order=0) // приоритет на нажатие
{
//--- установим координаты точек привязки, если они не заданы
    ChangeFiboLevelsEmptyPoints(timel,price1,time2,price2);
//---бросим значение ошибки
    ResetLastError();
//--- создадим "Уровни Фибоначчи" по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_FIBO,sub_window,timel,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Уровни Фибоначчи\"! Код ошибки = ",GetLastError())
        return(false);
    }
//--- установим цвет
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения объекта для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект

```

```

//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения объекта влево
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения объекта вправо
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Задает количество уровней и их параметры |+
//+-----+
bool FiboLevelsSet(int      levels,           // количество линий уровня
                    double   &values[],        // значения линий уровня
                    color    &colors[],        // цвет линий уровня
                    ENUM_LINE_STYLE &styles[], // стиль линий уровня
                    int      &widths[],         // толщина линий уровня
                    const long chart_ID=0,     // ID графика
                    const string name="FiboLevels") // имя объекта
{
//--- проверим размеры массивов
if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
   levels!=ArraySize(widths) || levels!=ArraySize(widths))
{
Print(__FUNCTION__,": длина массива не соответствует количеству уровней, ошибка");
return(false);
}
//--- установим количество уровней
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- установим свойства уровней в цикле
for(int i=0;i<levels;i++)
{
//--- значение уровня
ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
//--- цвет уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
//--- стиль уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
//--- толщина уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
//--- описание уровня
ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1));
}
}

```

```

//--- успешное выполнение
    return(true);
}
//+-----+
//| Перемещает точку привязки "Уровней Фибоначчи" |
//+-----+
bool FiboLevelsPointChange(const long    chart_ID=0,           // ID графика
                           const string name="FiboLevels", // имя объекта
                           const int     point_index=0,   // номер точки привязки
                           datetime    time=0,          // координата времени точки
                           double      price=0)        // координата цены точки пр

{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
    ResetLastError();
//--- переместим точку привязки
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Удаляет "Уровни Фибоначчи" |
//+-----+
bool FiboLevelsDelete(const long    chart_ID=0,           // ID графика
                      const string name="FiboLevels") // имя объекта
{
//---бросим значение ошибки
    ResetLastError();
//--- удалим объект
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить \"Уровни Фибоначчи\"! Код ошибки = ",GetLastError())
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Проверяет значения точек привязки "Уровней Фибоначчи" и для |

```

```

//| пустых значений устанавливает значения по умолчанию
//+-----+
void ChangeFiboLevelsEmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2)
{
//--- если время второй точки не задано, то она будет на текущем баре
if(!time2)
    time2=TimeCurrent();
//--- если цена второй точки не задана, то она будет иметь значение Bid
if(!price2)
    price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
if(!time1)
{
//--- массив для приема времени открытия 10 последних баров
datetime temp[10];
CopyTime(Symbol(),Period(),time2,10,temp);
//--- установим первую точку на 9 баров левее второй
time1=temp[0];
}
//--- если цена первой точки не задана, то сдвинем ее на 200 пунктов ниже второй
if(!price1)
    price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
}

//| Script program start function
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
{
Print("Ошибка! Некорректные значения входных параметров!");
return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки "Уровней Фибоначчи"
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
}

```

```

if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

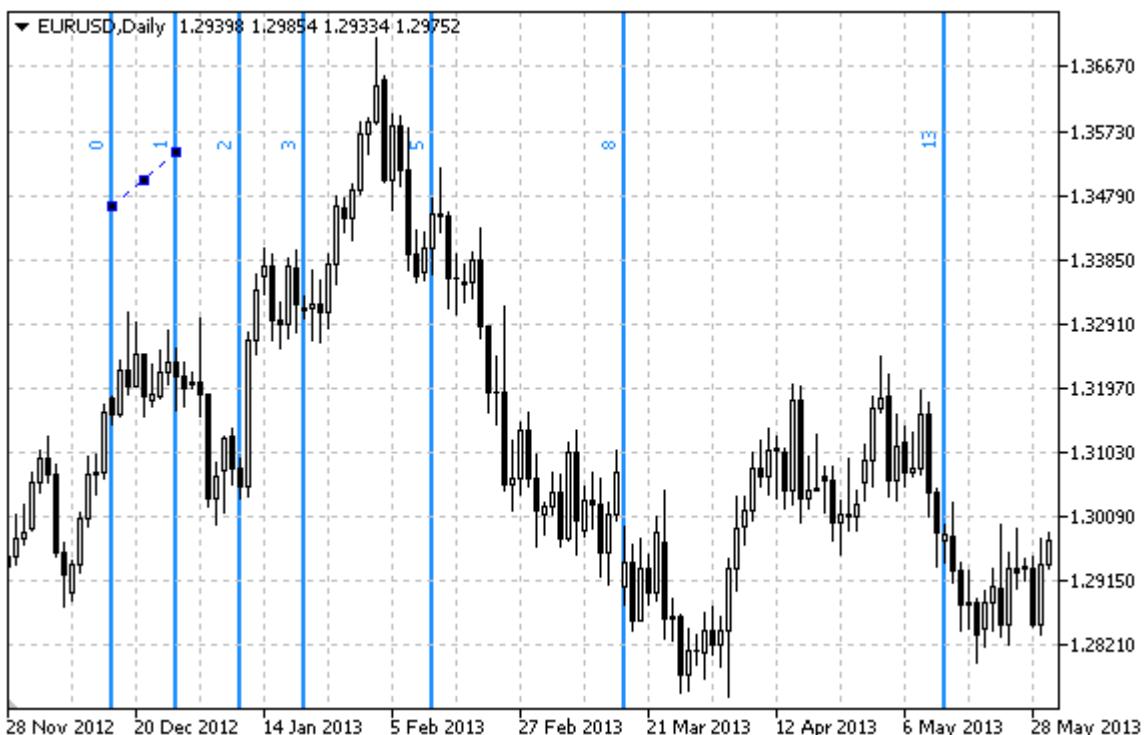
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования "Уровней Фибоначчи"
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
//--- создадим объект
if(!FiboLevelsCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
int v_steps=accuracy*2/5;
//--- перемещаем первую точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1>1)
        p1-=1;
    //--- сдвигаем точку
    if(!FiboLevelsPointChange(0,InpName,0,date[d1],price[p1]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
v_steps=accuracy*4/5;

```

```
//--- перемещаем вторую точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p2>1)
        p2-=1;
    //--- сдвигаем точку
    if(!FiboLevelsPointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим объект с графика
FiboLevelsDelete(0,InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_FIBOTIMES

Временные зоны Фибоначчи.



### Примечание

Для "Временных зон Фибоначчи" можно указать количество линий-уровней, их значения и цвет.

### Пример

Следующий скрипт создает и перемещает на графике "Временные зоны Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Временные зоны Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="FiboTimes";           // Имя объекта
input int              InpDate1=10;                  // Дата 1-ой точки в %
input int              InpPrice1=45;                 // Цена 1-ой точки в %
input int              InpDate2=20;                  // Дата 2-ой точки в %
input int              InpPrice2=55;                 // Цена 2-ой точки в %
input color            InpColor=clrRed;              // Цвет объекта
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линии
input int              InpWidth=2;                   // Толщина линии
input bool             InpBack=false;                // Объект на заднем плане

```

```

input bool           InpSelection=true;          // Выделить для перемещений
input bool           InpHidden=true;            // Скрыт в списке объектов
input long          InpZOrder=0;                // Приоритет на нажатие мышью
//+-----+
//| Создает "Временные зоны Фибоначчи" по заданным координатам |
//+-----+
bool FiboTimesCreate(const long             chart_ID=0,          // ID графика
                      const string         name="FiboTimes",    // имя объекта
                      const int            sub_window=0,      // номер подокна
                      datetime            timel=0,          // время первой точки
                      double               pricel=0,          // цена первой точки
                      datetime            time2=0,          // время второй точки
                      double               price2=0,          // цена второй точки
                      const color          clr=clrRed,        // цвет объекта
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии объекта
                      const int            width=1,          // толщина линии объекта
                      const bool           back=false,        // на заднем плане
                      const bool           selection=true,   // выделить для перемещения
                      const bool           hidden=true,       // скрыт в списке объектов
                      const long           z_order=0)        // приоритет на нажатие
{
//--- установим координаты точек привязки, если они не заданы
    ChangeFiboTimesEmptyPoints(timel,pricel,time2,price2);
//---бросим значение ошибки
    ResetLastError();
//--- создадим "Временные зоны Фибоначчи" по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_FIBOTIMES,sub_window,timel,pricel,time2,price2))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Временные зоны Фибоначчи\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения объекта для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Задает количество уровней и их параметры |
//+-----+

bool FiboTimesLevelsSet(int          levels,           // количество линий уровня
                        double        &values[],      // значения линий уровня
                        color         &colors[],     // цвет линий уровня
                        ENUM_LINE_STYLE &styles[],   // стиль линий уровня
                        int           &widths[],     // толщина линий уровня
                        const long    chart_ID=0,    // ID графика
                        const string   name="FiboTimes") // имя объекта

{
//--- проверим размеры массивов
if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
   levels!=ArraySize(widths) || levels!=ArraySize(widths))
{
Print(__FUNCTION__,"": длина массива не соответствует количеству уровней, ошибка");
return(false);
}

//--- установим количество уровней
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- установим свойства уровней в цикле
for(int i=0;i<levels;i++)
{
//--- значение уровня
ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
//--- цвет уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
//--- стиль уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
//--- толщина уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
//--- описание уровня
ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(values[i],1));
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки "Временных зон Фибоначчи" |
//+-----+

bool FiboTimesPointChange(const long    chart_ID=0,           // ID графика
                         const string  name="FiboTimes", // имя объекта
                         const int     point_index=0,    // номер точки привязки
                         datetime     time=0,          // координата времени точки привязки
                         double       price=0)         // координата цены точки привязки

```

```

{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
        ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет "Временные зоны Фибоначчи" |
//+-----+
bool FiboTimesDelete(const long    chart_ID=0,           // ID графика
                     const string name="FiboTimes") // имя объекта
{
//--- сбросим значение ошибки
ResetLastError();
//--- удалим объект
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
        ": не удалось удалить \"Временные зоны Фибоначчи\"! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Проверяет значения точек привязки "Временных зон Фибоначчи" и |
//| для пустых значений устанавливает значения по умолчанию |
//+-----+
void ChangeFiboTimesEmptyPoints(datetime &time1,double &price1,
                                 datetime &time2,double &price2)
{
//--- если время первой точки не задано, то она будет на текущем баре
if(!time1)
    time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
if(!price1)
    price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

```

```

//--- если время второй точки не задано, то она лежит на 2 бара левее второй
if(!time2)
{
    //--- массив для приема времени открытия 3 последних баров
    datetime temp[3];
    CopyTime(Symbol(),Period(),time1,3,temp);
    //--- установим первую точку на 2 бара левее второй
    time2=temp[0];
}

//--- если цена второй точки не задана, то она совпадает с ценой первой точки
if(!price2)
    price2=price1;
}

//-----+
//| Script program start function |+
//-----+
void OnStart()
{
    //--- проверим входные параметры на корректность
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
       InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }

    //--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

    //--- размер массива price
    int accuracy=1000;

    //--- массивы для хранения значений дат и цен, которые будут использованы
    //--- для установки и изменения координат точек привязки "Временных зон Фибоначчи"
    datetime date[];
    double price[];

    //--- выделение памяти
    ArrayResize(date,bars);
    ArrayResize(price,accuracy);

    //--- заполним массив дат
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
        return;
    }

    //--- заполним массив цен
    //--- найдем максимальное и минимальное значение графика
    double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
    double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

    //--- определим шаг изменения цены и заполним массив
    double step=(max_price-min_price)/accuracy;
}

```

```

for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования "Временных зон Фибоначчи"
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
//--- создадим объект
if(!FiboTimesCreate(0, InpName, 0, date[d1], price[p1], date[d2], price[p2],
    InpColor, InpStyle, InpWidth, InpBack, InpSelection, InpHidden, InpZOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
int h_steps=bars*2/5;
//--- перемещаем вторую точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d2<bars-1)
        d2+=1;
    //--- сдвигаем точку
    if(!FiboTimesPointChange(0, InpName, 1, date[d2], price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
h_steps=bars*3/5;
//--- перемещаем первую точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d1<bars-1)
        d1+=1;
    //--- сдвигаем точку
    if(!FiboTimesPointChange(0, InpName, 0, date[d1], price[p1]))
        return;
}

```

```
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}

//--- задержка в 1 секунду
Sleep(1000);
//--- удалим объект с графика
FiboTimesDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_FIBOFAN

Веер Фибоначчи.



### Примечание

Для "Веера Фибоначчи" можно указать количество линий-уровней, их значения и цвет.

### Пример

Следующий скрипт создает и перемещает на графике "Веер Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Веер Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="FiboFan";           // Имя веера
input int         InpDate1=10;                  // Дата 1-ой точки в %
input int         InpPrice1=25;                 // Цена 1-ой точки в %
input int         InpDate2=30;                  // Дата 2-ой точки в %
input int         InpPrice2=50;                 // Цена 2-ой точки в %
input color        InpColor=clrRed;              // Цвет линии веера
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линии
input int         InpWidth=2;                   // Толщина линии

```

```

input bool           InpBack=false;          // Объект на заднем плане
input bool           InpSelection=true;       // Выделить для перемещений
input bool           InpHidden=true;         // Скрыт в списке объектов
input long           InpZOrder=0;            // Приоритет на нажатие мышью
//+-----+
//| Создает "Веер Фибоначчи" по заданным координатам |
//+-----+
bool FiboFanCreate(const long           chart_ID=0,          // ID графика
                    const string        name="FiboFan",      // имя веера
                    const int            sub_window=0,       // номер подокна
                    datetime            time1=0,          // время первой точки
                    double               price1=0,          // цена первой точки
                    datetime            time2=0,          // время второй точки
                    double               price2=0,          // цена второй точки
                    const color          clr=clrRed,        // цвет линии веера
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии веера
                    const int             width=1,          // толщина линии веера
                    const bool            back=false,        // на заднем плане
                    const bool            selection=true,    // выделить для перемещений
                    const bool            hidden=true,       // скрыт в списке объектов
                    const long            z_order=0)        // приоритет на нажатие мышью
{
//--- установим координаты точек привязки, если они не заданы
    ChangeFiboFanEmptyPoints(time1,price1,time2,price2);
//---бросим значение ошибки
    ResetLastError();
//--- создадим "Веер Фибоначчи" по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_FIBOFAN,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Веер Фибоначчи\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения веера для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

```

```

//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Задает количество уровней и их параметры |
//+-----+

bool FiboFanLevelsSet(int      levels,           // количество линий уровня
                      double   &values[],        // значения линий уровня
                      color    &colors[],        // цвет линий уровня
                      ENUM_LINE_STYLE &styles[], // стиль линий уровня
                      int      &widths[],        // толщина линий уровня
                      const long chart_ID=0,     // ID графика
                      const string name="FiboFan") // имя веера

{
//--- проверим размеры массивов
if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
   levels!=ArraySize(widths) || levels!=ArraySize(widths))
{
Print(__FUNCTION__,"": длина массива не соответствует количеству уровней, ошибка!
return(false);
}

//--- установим количество уровней
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);

//--- установим свойства уровней в цикле
for(int i=0;i<levels;i++)
{
//--- значение уровня
ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
//--- цвет уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
//--- стиль уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
//--- толщина уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
//--- описание уровня
ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i]),
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки "Веера Фибоначчи" |
//+-----+

bool FiboFanPointChange(const long  chart_ID=0,       // ID графика
                       const string name="FiboFan", // имя веера
                       const int    point_index=0, // номер точки привязки
                       datetime   time=0,         // координата времени точки привязки

```

```

        double      price=0)          // координата цены точки привязки
    {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
    ResetLastError();
//--- переместим точку привязки
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

//+-----+
//| Удаляет "Веер Фибоначчи" |
//+-----+
bool FiboFanDelete(const long    chart_ID=0,      // ID графика
                   const string name="FiboFan") // имя веера
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим веер
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить \"Веер Фибоначчи\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

//+-----+
//| Проверяет значения точек привязки "Веера Фибоначчи" и для |
//| пустых значений устанавливает значения по умолчанию |
//+-----+
void ChangeFiboFanEmptyPoints(datetime &time1,double &price1,
                               datetime &time2,double &price2)
{
//--- если время второй точки не задано, то она будет на текущем баре
    if(!time2)
        time2=TimeCurrent();
//--- если цена второй точки не задана, то она будет иметь значение Bid
    if(!price2)

```

```

price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
if(!time1)
{
    //--- массив для приема времени открытия 10 последних баров
    datetime temp[10];
    CopyTime(Symbol(),Period(),time2,10,temp);
    //--- установим первую точку на 9 баров левее второй
    time1=temp[0];
}
//--- если цена первой точки не задана, то сдвинем ее на 200 пунктов ниже второй
if(!price1)
    price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
}

//-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки "Веера Фибоначчи"
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив

```

```

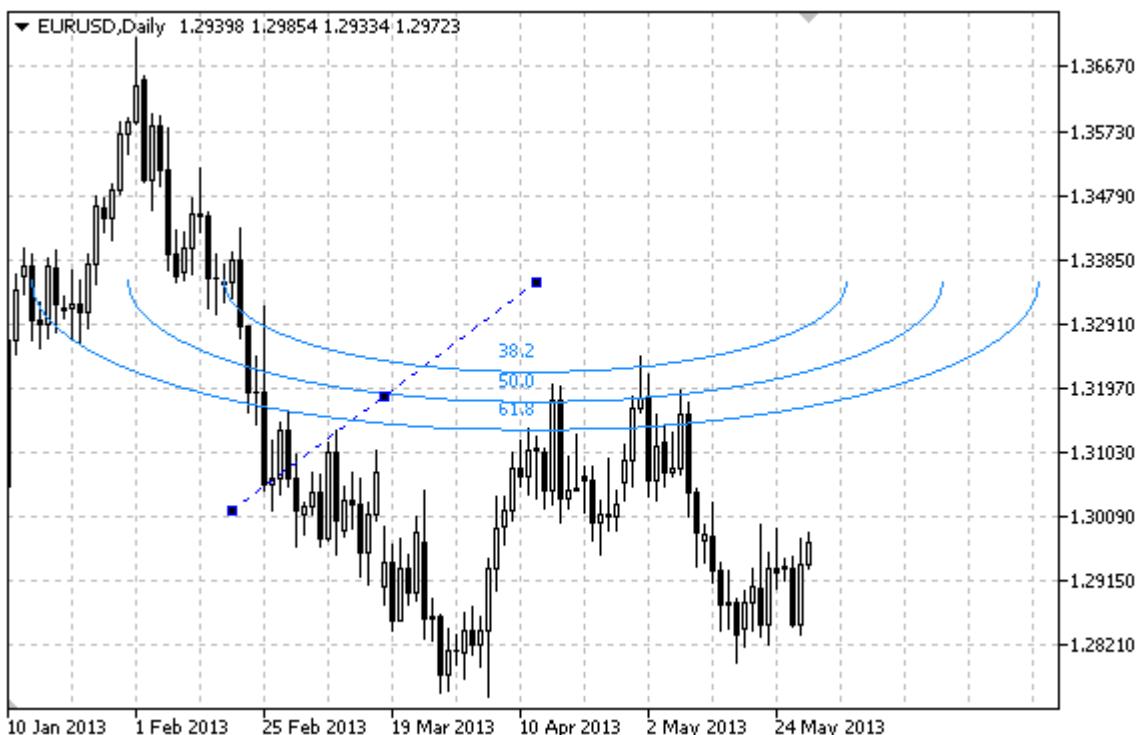
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования "Веера Фибоначчи"
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
//--- создадим объект
if(!FiboFanCreate(0, InpName, 0, date[d1], price[p1], date[d2], price[p2],
    InpColor, InpStyle, InpWidth, InpBack, InpSelection, InpHidden, InpZOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки веера
//--- счетчик цикла
int v_steps=accuracy/2;
//--- перемещаем первую точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1<accuracy-1)
        p1+=1;
    //--- сдвигаем точку
    if(!FiboFanPointChange(0, InpName, 0, date[d1], price[p1]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
int h_steps=bars/4;
//--- перемещаем вторую точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d2<bars-1)
        d2+=1;
    //--- сдвигаем точку
    if(!FiboFanPointChange(0, InpName, 1, date[d2], price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
}

```

```
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим объект с графика
FiboFanDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
}
```

## OBJ\_FIBOARC

Дуги Фибоначчи.



### Примечание

Для "Дуги Фибоначчи" можно указать режим отображения всего эллипса целиком. Радиус кривизны линий можно задать изменяя масштаб и координаты точек привязки.

Также можно указать количество линий-уровней, их значения и цвет.

### Пример

Следующий скрипт создает и перемещает на графике "Дуги Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Дуги Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="FiboArc";           // Имя объекта
input int         InpDate1=25;                  // Дата 1-ой точки в %
input int         InpPrice1=25;                 // Цена 1-ой точки в %
input int         InpDate2=35;                  // Дата 2-ой точки в %
input int         InpPrice2=55;                 // Цена 2-ой точки в %
input double      InpScale=3.0;                // Масштаб
```

```

input bool           InpFullEllipse=true;          // Форма дуг
input color         InpColor=clrRed;              // Цвет линии
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линии
input int            InpWidth=2;                  // Толщина линии
input bool           InpBack=false;               // Объект на заднем плане
input bool           InpSelection=true;            // Выделить для перемещений
input bool           InpHidden=true;               // Скрыт в списке объектов
input long           InpZOrder=0;                 // Приоритет на нажатие мышью
//+-----+
//| Создает "Дуги Фибоначчи" по заданным координатам |
//+-----+
bool FiboArcCreate(const long           chart_ID=0,           // ID графика
                    const string        name="FiboArc",       // имя объекта
                    const int            sub_window=0,        // номер подокна
                    datetime            time1=0,           // время первой точки
                    double               price1=0,           // цена первой точки
                    datetime            time2=0,           // время второй точки
                    double               price2=0,           // цена второй точки
                    const double         scale=1.0,          // масштаб
                    const bool            full_ellipse=false, // форма дуг
                    const color           clr=clrRed,          // цвет линии
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                    const int             width=1,            // толщина линии
                    const bool            back=false,          // на заднем плане
                    const bool            selection=true,      // выделить для перемещений
                    const bool            hidden=true,          // скрыт в списке объектов
                    const long            z_order=0)          // приоритет на нажатие мышью
{
//--- установим координаты точек привязки, если они не заданы
    ChangeFiboArcEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим "Дуги Фибоначчи" по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_FIBOARC,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Дуги Фибоначчи\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим масштаб
    ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- установим отображение дуг в виде полного эллипса (true) или половины (false)
    ObjectSetInteger(chart_ID,name,OBJPROP_ELLIPSE,full_ellipse);
//--- установим цвет
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);

```

```

//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);

//--- включим (true) или отключим (false) режим выделения дуг для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

//--- успешное выполнение
return(true);
}

//+-----+
//| Задает количество уровней и их параметры |+
//+-----+

bool FiboArcLevelsSet(int      levels,           // количество линий уровня
                      double    &values[],        // значения линий уровня
                      color     &colors[],       // цвет линий уровня
                      ENUM_LINE_STYLE &styles[], // стиль линий уровня
                      int       &widths[],        // толщина линий уровня
                      const long chart_ID=0,     // ID графика
                      const string name="FiboArc") // имя объекта
{
    //--- проверим размеры массивов
    if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
       levels!=ArraySize(widths) || levels!=ArraySize(widths))
    {
        Print(__FUNCTION__,": длина массива не соответствует количеству уровней, ошибка");
        return(false);
    }

    //--- установим количество уровней
    ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);

    //--- установим свойства уровней в цикле
    for(int i=0;i<levels;i++)
    {
        //--- значение уровня
        ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
        //--- цвет уровня
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
        //--- стиль уровня
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
        //--- толщина уровня
        ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
        //--- описание уровня
        ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i]));
    }
}

```

```

//--- успешное выполнение
    return(true);
}
//+-----+
//| Перемещает точку привязки "Дуг Фибоначчи" |
//+-----+
bool FiboArcPointChange(const long    chart_ID=0,          // ID графика
                        const string name="FiboArc", // имя объекта
                        const int     point_index=0, // номер точки привязки
                        datetime    time=0,        // координата времени точки привязки
                        double       price=0)      // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
    ResetLastError();
//--- переместим точку привязки
    if(!ObjectMove(chart_ID,name,point_index,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Удаляет "Дуги Фибоначчи" |
//+-----+
bool FiboArcDelete(const long    chart_ID=0,          // ID графика
                   const string name="FiboArc") // имя объекта
{
//---бросим значение ошибки
    ResetLastError();
//--- удалим объект
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить \"Дуги Фибоначчи\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Проверяет значения точек привязки "Дуг Фибоначчи" и для пустых |

```

```

//| значений устанавливает значения по умолчанию
//+-----+
void ChangeFiboArcEmptyPoints(datetime &time1,double &price1,
                               datetime &time2,double &price2)
{
//--- если время второй точки не задано, то она будет на текущем баре
if(!time2)
    time2=TimeCurrent();
//--- если цена второй точки не задана, то она будет иметь значение Bid
if(!price2)
    price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
if(!time1)
{
//--- массив для приема времени открытия 10 последних баров
datetime temp[10];
CopyTime(Symbol(),Period(),time2,10,temp);
//--- установим первую точку на 9 баров левее второй
time1=temp[0];
}
//--- если цена первой точки не задана, то сдвинем ее на 300 пунктов ниже второй
if(!price1)
    price1=price2-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
}

//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
{
Print("Ошибка! Некорректные значения входных параметров!");
return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки "Дуг Фибоначчи"
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
}

```

```

if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования "Дуг Фибоначчи"
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
//--- создадим объект
if(!FiboArcCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpScale,
    InpFullEllipse,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
int v_steps=accuracy/5;
//--- перемещаем первую точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1<accuracy-1)
        p1+=1;
    //--- сдвигаем точку
    if(!FiboArcPointChange(0,InpName,0,date[d1],price[p1]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
int h_steps=bars/5;

```

```
//--- перемещаем вторую точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d2<bars-1)
        d2+=1;
    //--- сдвигаем точку
    if(!FiboArcPointChange(0,IInpName,1,date[d2],price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим объект с графика
FiboArcDelete(0,IInpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_FIBOCHANNEL

Канал Фибоначчи.



### Примечание

Для "Канала Фибоначчи" можно указать режим продолжения его отображения вправо и/или влево (свойства [OBJPROP\\_RAY\\_RIGHT](#) и [OBJPROP\\_RAY\\_LEFT](#) соответственно).

Также можно указать количество линий-уровней, их значения и цвет.

### Пример

Следующий скрипт создает и перемещает на графике "Канал Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Канал Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="FiboChannel";           // Имя канала
input int         InpDate1=20;                      // Дата 1-ой точки в %
input int         InpPrice1=10;                     // Цена 1-ой точки в %
input int         InpDate2=60;                     // Дата 2-ой точки в %
input int         InpPrice2=30;                     // Цена 2-ой точки в %
input int         InpDate3=20;                     // Дата 3-ей точки в %
```

```

input int          InpPrice3=25;           // Цена 3-ей точки в %
input color        InpColor=clrRed;        // Цвет канала
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий канала
input int          InpWidth=2;            // Толщина линий канала
input bool         InpBack=false;         // Канал на заднем плане
input bool         InpSelection=true;       // Выделить для перемещений
input bool         InpRayLeft=false;        // Продолжение канала влево
input bool         InpRayRight=false;       // Продолжение канала вправо
input bool         InpHidden=true;          // Скрыт в списке объектов
input long         InpZOrder=0;            // Приоритет на нажатие мышью
//+-----+
//| Создает "Канал Фибоначчи" по заданным координатам |+
//+-----+
bool FiboChannelCreate(const long           chart_ID=0,           // ID графика
                       const string        name="FiboChannel", // имя канала
                       const int           sub_window=0,      // номер подокна
                       const datetime       time1=0,          // время первой точки
                       const double         price1=0,         // цена первой точки
                       const datetime       time2=0,          // время второй точки
                       const double         price2=0,         // цена второй точки
                       const datetime       time3=0,          // время третьей точки
                       const double         price3=0,         // цена третьей точки
                       const color          clr=clrRed,        // цвет канала
                       const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий канала
                       const int           width=1,          // толщина линий канала
                       const bool          back=false,        // на заднем плане
                       const bool          selection=true,    // выделить для перемещения
                       const bool          ray_left=false,     // продолжение канала влево
                       const bool          ray_right=false,   // продолжение канала вправо
                       const bool          hidden=true,       // скрыт в списке объектов
                       const long          z_order=0)        // приоритет на нажатие мышью
{
//--- установим координаты точек привязки, если они не заданы
    ChangeFiboChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//---бросим значение ошибки
    ResetLastError();
//--- создадим канал по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_FIBOCHANNEL,sub_window,time1,price1,time2,price2,
                     time3,price3))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Канал Фибоначчи\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет канала
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий канала
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий канала
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
}

```

```

//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);

//--- включим (true) или отключим (false) режим выделения канала для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- включим (true) или отключим (false) режим продолжения отображения канала влево
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения канала вправо
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);

//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

//--- успешное выполнение
return(true);
}

//-----+
//| Задает количество уровней и их параметры |
//-----+
bool FiboChannelLevelsSet(int           levels,          // количество линий уровня
                           double        &values[],       // значения линий уровня
                           color         &colors[],      // цвет линий уровня
                           ENUM_LINE_STYLE &styles[],    // стиль линий уровня
                           int            &widths[],      // толщина линий уровня
                           const long     chart_ID=0,    // ID графика
                           const string   name="FiboChannel") // имя объекта

{
//--- проверим размеры массивов
if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
   levels!=ArraySize(widths) || levels!=ArraySize(widths))
{
Print(__FUNCTION__,"": длина массива не соответствует количеству уровней, ошибка");
return(false);
}

//--- установим количество уровней
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);

//--- установим свойства уровней в цикле
for(int i=0;i<levels;i++)
{
//--- значение уровня
ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
//--- цвет уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
//--- стиль уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
//--- толщина уровня
}
}

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
//--- описание уровня
ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],1));
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки "Канала Фибоначчи" |
//+-----+
bool FiboChannelPointChange(const long    chart_ID=0,           // ID графика
                           const string name="FiboChannel", // имя канала
                           const int     point_index=0,      // номер точки привязки
                           datetime    time=0,             // координата времени точки
                           double       price=0)          // координата цены точки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет канал |
//+-----+
bool FiboChannelDelete(const long    chart_ID=0,           // ID графика
                      const string name="FiboChannel") // имя канала
{
//---бросим значение ошибки
ResetLastError();
//---удалим канал
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": не удалось удалить \"Канал Фибоначчи\"! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение

```

```

        return(true);
    }

//+-----+
//| Проверяет значения точек привязки "Канала Фибоначчи" и для           |
//| пустых значений устанавливает значения по умолчанию                   |
//+-----+
void ChangeFiboChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                                    double &price2,datetime &time3,double &price3)
{
    //--- если время второй (правой) точки не задано, то она будет на текущем баре
    if(!time2)
        time2=TimeCurrent();

    //--- если цена второй точки не задана, то она будет иметь значение Bid
    if(!price2)
        price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);

    //--- если время первой (левой) точки не задано, то она лежит на 9 баров левее второй
    if(!time1)
    {
        //--- массив для приема времени открытия 10 последних баров
        datetime temp[10];
        CopyTime(Symbol(),Period(),time2,10,temp);
        //--- установим первую точку на 9 баров левее второй
        time1=temp[0];
    }

    //--- если цена первой точки не задана, то сдвинем ее на 300 пунктов выше второй
    if(!price1)
        price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);

    //--- если время третьей точки не задано, то оно совпадает с временем первой точки
    if(!time3)
        time3=time1;

    //--- если цена третьей точки не задана, то она совпадает с ценой второй точки
    if(!price3)
        price3=price2;
}

//+-----+
//| Script program start function                                         |
//+-----+
void OnStart()
{
    //--- проверим входные параметры на корректность
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
       InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
       InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }

    //--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
}

```

```

//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки канала
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования канала
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int d3=InpDate3*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
int p3=InpPrice3*(accuracy-1)/100;
//--- создадим "Канал Фибоначчи"
if(!FiboChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],p
    InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHider
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки канала
//--- счетчик цикла
int h_steps=bars/10;
//--- перемещаем первую точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d1>1)
        d1-=1;
}

```

```

//--- сдвигаем точку
if (!FiboChannelPointChange(0, InpName, 0, date[d1], price[p1]))
    return;
//--- проверим факт принудительного завершения скрипта
if (IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}

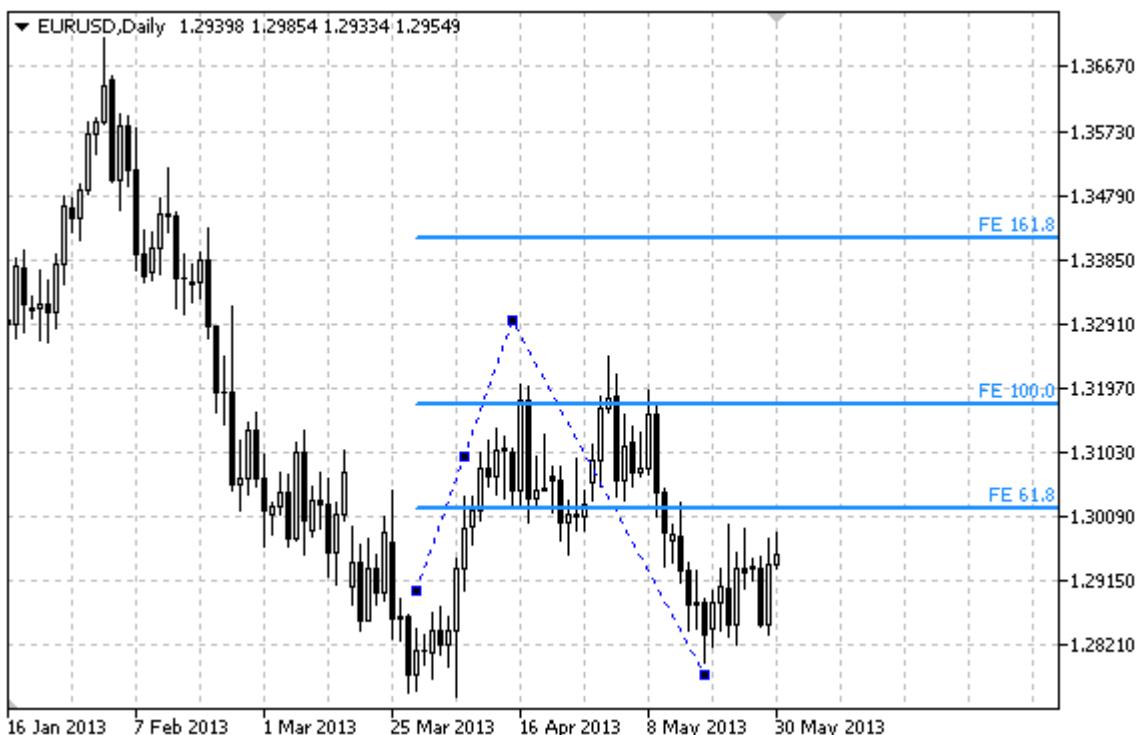
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
int v_steps=accuracy/10;
//--- перемещаем вторую точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p2>1)
        p2-=1;
    //--- сдвигаем точку
    if (!FiboChannelPointChange(0, InpName, 1, date[d2], price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if (IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
v_steps=accuracy/15;
//--- перемещаем третью точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p3<accuracy-1)
        p3+=1;
    //--- сдвигаем точку
    if (!FiboChannelPointChange(0, InpName, 2, date[d3], price[p3]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if (IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}

```

```
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим канал с графика
FiboChannelDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
}
```

## OBJ\_EXPANSION

Расширение Фибоначчи.



### Примечание

Для "Расширения Фибоначчи" можно указать режим продолжения отображения вправо и/или влево (свойства [OBJPROP\\_RAY\\_RIGHT](#) и [OBJPROP\\_RAY\\_LEFT](#) соответственно).

Также можно указать количество линий-уровней, их значения и цвет.

### Пример

Следующий скрипт создает и перемещает на графике "Расширение Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Расширение Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="FiboExpansion";    // Имя объекта
input int         InpDate1=10;                  // Дата 1-ой точки в %
input int         InpPrice1=55;                 // Цена 1-ой точки в %
input int         InpDate2=30;                  // Дата 2-ой точки в %
input int         InpPrice2=10;                 // Цена 2-ой точки в %
input int         InpDate3=80;                  // Дата 3-ей точки в %
```

```

input int          InpPrice3=75;           // Цена 3-ей точки в %
input color        InpColor=clrRed;        // Цвет объекта
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий
input int          InpWidth=2;            // Толщина линий
input bool         InpBack=false;         // Объект на заднем плане
input bool         InpSelection=true;       // Выделить для перемещений
input bool         InpRayLeft=false;        // Продолжение объекта влево
input bool         InpRayRight=false;       // Продолжение объекта вправо
input bool         InpHidden=true;          // Скрыт в списке объектов
input long         InpZOrder=0;            // Приоритет на нажатие мышью
//+-----+
//| Создает "Расширение Фибоначчи" по заданным координатам |+
//+-----+
bool FiboExpansionCreate(const long          chart_ID=0,           // ID графика
                         const string        name="FiboExpansion", // имя канала
                         const int           sub_window=0,      // номер подокна
                         const datetime      time1=0,          // время первой т
                         const double        price1=0,         // цена первой т
                         const datetime      time2=0,          // время второй т
                         const double        price2=0,         // цена второй т
                         const datetime      time3=0,          // время третьей т
                         const double        price3=0,         // цена третьей т
                         const color         clr=clrRed,        // цвет объекта
                         const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий
                         const int           width=1,          // толщина линий
                         const bool          back=false,        // на заднем плане
                         const bool          selection=true,    // выделить для г
                         const bool          ray_left=false,   // продолжение об
                         const bool          ray_right=false,  // продолжение об
                         const bool          hidden=true,       // скрыт в списке
                         const long          z_order=0)        // приоритет на р
{
//--- установим координаты точек привязки, если они не заданы
    ChangeFiboExpansionEmptyPoints(time1,price1,time2,price2,time3,price3);
//---бросим значение ошибки
    ResetLastError();
//--- создадим "Расширение Фибоначчи" по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_EXPANSION,sub_window,time1,price1,time2,price2,
                     time3,price3))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Расширение Фибоначчи\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет объекта
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);

```

```

//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);

//--- включим (true) или отключим (false) режим выделения объекта для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- включим (true) или отключим (false) режим продолжения отображения объекта влево
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения объекта вправо
ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);

//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

//--- успешное выполнение
return(true);
}

//+-----+
//| Задает количество уровней и их параметры |+
//+-----+
bool FiboExpansionLevelsSet(int levels, // количество линий
                            double &values[], // значения линий уровня
                            color &colors[], // цвет линий уровня
                            ENUM_LINE_STYLE &styles[], // стиль линий уровня
                            int &widths[], // толщина линий уровня
                            const long chart_ID=0, // ID графика
                            const string name="FiboExpansion") // имя объекта

{
//--- проверим размеры массивов
if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
   levels!=ArraySize(widths) || levels!=ArraySize(widths))
{
Print(__FUNCTION__,"": длина массива не соответствует количеству уровней, ошибка!");
return(false);
}

//--- установим количество уровней
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);

//--- установим свойства уровней в цикле
for(int i=0;i<levels;i++)
{
//--- значение уровня
ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
//--- цвет уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
//--- стиль уровня
ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
//--- толщина уровня
}
}

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
//--- описание уровня
ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,"FE "+DoubleToString(100*value));
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки "Расширения Фибоначчи" |
//+-----+
bool FiboExpansionPointChange(const long    chart_ID=0,           // ID графика
                               const string  name="FiboExpansion", // имя объекта
                               const int     point_index=0,        // номер точки привязки
                               datetime    time=0,              // координата времени
                               double       price=0)            // координата цены точки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет "Расширение Фибоначчи" |
//+-----+
bool FiboExpansionDelete(const long    chart_ID=0,           // ID графика
                        const string  name="FiboExpansion") // имя объекта
{
//---бросим значение ошибки
ResetLastError();
//---удалим объект
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": не удалось удалить \"Расширение Фибоначчи\"! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
}

```

```

        return(true);
    }

//+-----+
//| Проверяет значения точек привязки "Расширения Фибоначчи" и для |
//| пустых значений устанавливает значения по умолчанию           |
//+-----+
void ChangeFiboExpansionEmptyPoints(datetime &time1,double &price1,datetime &time2,
                                      double &price2,datetime &time3,double &price3)
{
    //--- если время третьей (правой) точки не задано, то она будет на текущем баре
    if(!time3)
        time3=TimeCurrent();
    //--- если цена третьей точки не задана, то она будет иметь значение Bid
    if(!price3)
        price3=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    //--- если время первой (левой) точки не задано, то она лежит на 9 баров левее третьей
    //--- массив для приема времени открытия 10 последних баров
    datetime temp[];
    ArrayResize(temp,10);
    if(!time1)
    {
        CopyTime(Symbol(),Period(),time3,10,temp);
        //--- установим первую точку на 9 баров левее второй
        time1=temp[0];
    }
    //--- если цена первой точки не задана, то она совпадает с ценой третьей точки
    if(!price1)
        price1=price3;
    //--- если время второй точки не задано, то она лежит на 7 баров левее третьей
    if(!time2)
        time2=temp[2];
    //--- если цена второй точки не задана, сдвинем ее на 250 пунктов ниже первой
    if(!price2)
        price2=price1-250*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
    }

//+-----+
//| Script program start function                                |
//+-----+
void OnStart()
{
    //--- проверим входные параметры на корректность
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
       InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
       InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }
    //--- количество видимых баров в окне графика
}

```

```

int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки объекта
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования "Расширения Фибоначчи"
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int d3=InpDate3*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
int p3=InpPrice3*(accuracy-1)/100;
//--- создадим "Расширение Фибоначчи"
if(!FiboExpansionCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],
    InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHeader))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
int v_steps=accuracy/10;
//--- перемещаем первую точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1>1)

```

```

p1-=1;
//--- сдвигаем точку
if(!FiboExpansionPointChange(0, InpName, 0, date[d1], price[p1]))
    return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
v_steps=accuracy/2;
//--- перемещаем третью точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p3>1)
        p3-=1;
    //--- сдвигаем точку
    if(!FiboExpansionPointChange(0, InpName, 2, date[d3], price[p3]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
v_steps=accuracy*4/5;
//--- перемещаем вторую точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p2<accuracy-1)
        p2+=1;
    //--- сдвигаем точку
    if(!FiboExpansionPointChange(0, InpName, 1, date[d2], price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду

```

```
Sleep(1000);
//--- удалим объект с графика
FiboExpansionDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
}
```

## OBJ\_ELLIOTWAVE5

Импульсная волна Эллиота.



### Примечание

Для "Импульсной волны Эллиота" можно включить/отключить режим соединения точек линиями (свойство `OBJPROP_DRAWLINES`), а также установить уровень волновой разметки (из перечисления `ENUM_ELLIOT_WAVE_DEGREE`).

### Пример

Следующий скрипт создает и перемещает на графике "Импульсную волну Эллиота". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Импульсная волна Эллиота\"."
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="ElliotWave5";      // Имя объекта
input int                InpDate1=10;                 // Дата 1-ой точки в %
input int                InpPrice1=90;                // Цена 1-ой точки в %
input int                InpDate2=20;                 // Дата 2-ой точки в %
input int                InpPrice2=40;                // Цена 2-ой точки в %
input int                InpDate3=30;                 // Дата 3-ей точки в %

```

```

input int                     InpPrice3=60;           // Цена 3-ей точки в %
input int                     InpDate4=40;           // Дата 4-ой точки в %
input int                     InpPrice4=10;           // Цена 4-ой точки в %
input int                     InpDate5=60;           // Дата 5-ой точки в %
input int                     InpPrice5=40;           // Цена 5-ой точки в %
input ENUM_ELLIOT_WAVE_DEGREE InpDegree=ELLIOTT_MINOR; // Уровень
input bool                    InpDrawLines=true;        // Отображение линий
input color                   InpColor=clrRed;         // Цвет линий
input ENUM_LINE_STYLE          InpStyle=STYLE_DASH;       // Стиль линий
input int                     InpWidth=2;            // Толщина линий
input bool                    InpBack=false;          // Объект на заднем плане
input bool                    InpSelection=true;        // Выделить для перемещений
input bool                    InpHidden=true;          // Скрыт в списке объектов
input long                    InpZOrder=0;            // Приоритет на нажатие мышью
//----------------------------------------------------------------------------------------------------------------+
//| Создает "Импульсную волну Эллиота" по заданным координатам |+
//----------------------------------------------------------------------------------------------------------------+
bool ElliotWave5Create(const long                  chart_ID=0,           // ID гра
                       const string              name="ElliotWave5",      // имя волны
                       const int                 sub_window=0,        // номер подокна
                       const datetime             time1=0,            // время 1
                       const double               price1=0,           // цена 1
                       const datetime             time2=0,            // время 2
                       const double               price2=0,           // цена 2
                       const datetime             time3=0,            // время 3
                       const double               price3=0,           // цена 3
                       const datetime             time4=0,            // время 4
                       const double               price4=0,           // цена 4
                       const datetime             time5=0,            // время 5
                       const double               price5=0,           // цена 5
                       const ENUM_ELLIOT_WAVE_DEGREE degree=ELLIOTT_MINUETTE, // степень
                       const bool                draw_lines=true,        // отображение
                       const color                clr=clrRed,          // цвет
                       const ENUM_LINE_STYLE      style=STYLE_SOLID,       // стиль
                       const int                 width=1,             // толщина
                       const bool                back=false,          // на задний план
                       const bool                selection=true,        // выделение
                       const bool                hidden=true,          // скрытый
                       const long                z_order=0)           // приоритет
{
//--- установим координаты точек привязки, если они не заданы
    ChangeElliotWave5EmptyPoints(time1,price1,time2,price2,time3,price3,time4,price4,time5,price5);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим "Импульсную волну Эллиота" по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_ELLIOTWAVE5,sub_window,time1,price1,time2,price2,
                     price3,time4,price4,time5,price5))
    {
        Print(__FUNCTION__, ...

```

```

        ": не удалось создать \"Импульсную волну Эллиота\"! Код ошибки = ", GetLastError();
    return(false);
}

//--- установим степень (размер волны)
ObjectSetInteger(chart_ID,name,OBJPROP_DEGREE,degree);
//--- включим (true) или отключим (false) режим отображения линий
ObjectSetInteger(chart_ID,name,OBJPROP_DRAWLINES,draw_lines);
//--- установим цвет объекта
ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий
ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения объекта для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки "Импульсной волны Эллиота" |
//+-----+
bool ElliotWave5PointChange(const long    chart_ID=0,           // ID графика
                           const string name="ElliotWave5", // имя объекта
                           const int     point_index=0,      // номер точки привязки
                           datetime    time=0,             // координата времени точки
                           double      price=0)          // координата цены точки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
        ": не удалось переместить точку привязки! Код ошибки = ", GetLastError());
}

```

```

        return(false);
    }

//--- успешное выполнение
    return(true);
}

//+-----+
// | Удаляет "Импульсную волну Эллиота"
//+-----+
bool ElliotWave5Delete(const long chart_ID=0,           // ID графика
                       const string name="ElliotWave5") // имя объекта
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим объект
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить \"Импульсную волну Эллиота\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

//+-----+
// | Проверяет значения точек привязки "Импульсной волны Эллиота" и |
// | для пустых значений устанавливает значения по умолчанию          |
//+-----+
void ChangeElliotWave5EmptyPoints(datetime &time1,double &price1,
                                   datetime &time2,double &price2,
                                   datetime &time3,double &price3,
                                   datetime &time4,double &price4,
                                   datetime &time5,double &price5)
{
//--- массив для приема времени открытия 10 последних баров
    datetime temp[];
    ArrayResize(temp,10);
//--- получим данные
    CopyTime(Symbol(),Period(),TimeCurrent(),10,temp);
//--- получим величину одного пункта на текущем графике
    double point=SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время первой точки не задано, то она будет на 9 баре слева от последнего бара
    if(!time1)
        time1=temp[0];
//--- если цена первой точки не задана, то она будет иметь значение Bid
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она будет на 7 баре слева от последнего бара
    if(!time2)
        time2=temp[2];
}

```

```

//--- если цена второй точки не задана, сдвинем ее на 300 пунктов ниже первой
if(!price2)
    price2=price1-300*point;
//--- если время третьей точки не задано, то она будет на 5 баров слева от последнего
if(!time3)
    time3=temp[4];
//--- если цена третьей точки не задана, сдвинем ее на 250 пунктов ниже первой
if(!price3)
    price3=price1-250*point;
//--- если время четвертой точки не задано, то она будет на 3 бара слева от последнего
if(!time4)
    time4=temp[6];
//--- если цена четвертой точки не задана, сдвинем ее на 550 пунктов ниже первой
if(!price4)
    price4=price1-550*point;
//--- если время пятой точки не задано, то она будет на последнем баре
if(!time5)
    time5=temp[9];
//--- если цена пятой точки не задана, сдвинем ее на 450 пунктов ниже первой
if(!price5)
    price5=price1-450*point;
}

//-----+
//| Script program start function |
//-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
   InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100 ||
   InpDate4<0 || InpDate4>100 || InpPrice4<0 || InpPrice4>100 ||
   InpDate5<0 || InpDate5>100 || InpPrice5<0 || InpPrice5>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки объекта
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат

```

```

ResetLastError();

if(CopyTime(Symbol(), Period(), 0, bars, date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ", GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования "Импульсной волны Эллиота"
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int d3=InpDate3*(bars-1)/100;
int d4=InpDate4*(bars-1)/100;
int d5=InpDate5*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
int p3=InpPrice3*(accuracy-1)/100;
int p4=InpPrice4*(accuracy-1)/100;
int p5=InpPrice5*(accuracy-1)/100;
//--- создадим "Импульсную волну Эллиота"
if(!ElliotWave5Create(0, InpName, 0, date[d1], price[p1], date[d2], price[p2], date[d3], price[p3], date[d4], price[p4], date[d5], price[p5], InpDegree, InpDrawLines, InpColor, InpStyle, InpDash, InpWidth, InpBack, InpSelection, InpHidden, InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать точки привязки
//--- счетчик цикла
int v_steps=accuracy/5;
//--- перемещаем пятую точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p5<accuracy-1)
        p5+=1;
    //--- сдвигаем точку
    if(!ElliotWave5PointChange(0, InpName, 4, date[d5], price[p5]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
}

```

```

        return;
//--- перерисуем график
ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
v_steps=accuracy/5;
//--- перемещаем вторую и третью точки привязки
for(int i=0;i<v_steps;i++)
{
//--- возьмем следующие значения
if(p2<accuracy-1)
    p2+=1;
if(p3>1)
    p3-=1;
//--- сдвигаем точки
if(!ElliotWave5PointChange(0,InpName,1,date[d2],price[p2]))
    return;
if(!ElliotWave5PointChange(0,InpName,2,date[d3],price[p3]))
    return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
v_steps=accuracy*4/5;
//--- перемещаем первую и четвертую точки привязки
for(int i=0;i<v_steps;i++)
{
//--- возьмем следующие значения
if(p1>1)
    p1-=1;
if(p4<accuracy-1)
    p4+=1;
//--- сдвигаем точки
if(!ElliotWave5PointChange(0,InpName,0,date[d1],price[p1]))
    return;
if(!ElliotWave5PointChange(0,InpName,3,date[d4],price[p4]))
    return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
}

```

```
}

//--- задержка в 1 секунду
Sleep(1000);

//--- удалим объект с графика
ElliotWave5Delete(0, InpName);
ChartRedraw();

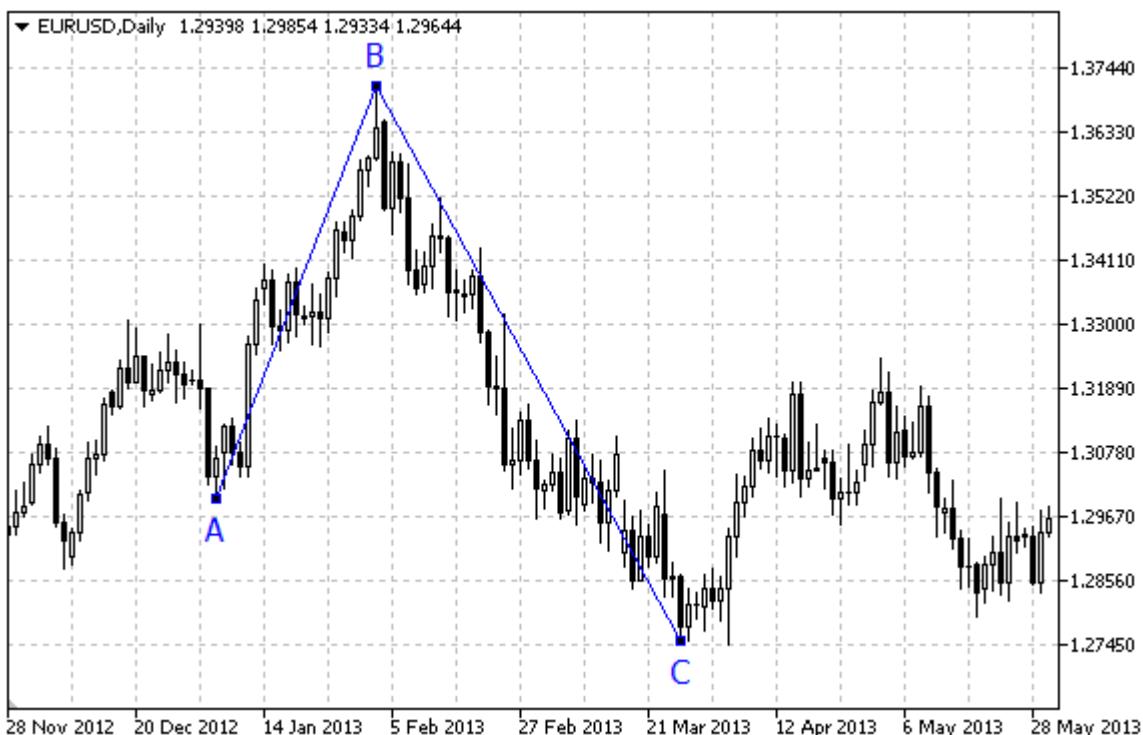
//--- задержка в 1 секунду
Sleep(1000);

//---

}
```

## OBJ\_ELLIOTWAVE3

Корректирующая волна Эллиота.



### Примечание

Для "Корректирующей волны Эллиота" можно включить/отключить режим соединения точек линиями (свойство [OBJPROP\\_DRAWLINES](#)), а также установить уровень волновой разметки (из перечисления [ENUM\\_ELLIOT\\_WAVE\\_DEGREE](#)).

### Пример

Следующий скрипт создает и перемещает на графике "Корректирующую волну Эллиота". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Корректирующая волна Эллиота"
#property description "Координаты точек привязки задаются в процентах от размеров окна"
#property description "графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="ElliotWave3";      // Имя объекта
input int                InpDate1=10;                 // Дата 1-ой точки в %
input int                InpPrice1=90;                 // Цена 1-ой точки в %
input int                InpDate2=30;                 // Дата 2-ой точки в %
input int                InpPrice2=10;                 // Цена 2-ой точки в %
input int                InpDate3=50;                 // Дата 3-ей точки в %

```

```

input int InpPrice3=40; // Цена 3-ей точки в %
input ENUM_ELLIOT_WAVE_DEGREE InpDegree=ELLIOTT_MINOR; // Уровень
input bool InpDrawLines=true; // Отображение линий
input color InpColor=clrRed; // Цвет линий
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линий
input int InpWidth=2; // Толщина линий
input bool InpBack=false; // Объект на заднем плане
input bool InpSelection=true; // Выделить для перемещений
input bool InpHidden=true; // Скрыт в списке объектов
input long InpZOrder=0; // Приоритет на нажатие мышью
//+-----+
//| Создает "Корректирующую волну Эллиота" по заданным координатам |
//+-----+
bool ElliotWave3Create(const long chart_ID=0, // ID графика
                       const string name="ElliotWave3", // имя объекта
                       const int sub_window=0, // номер окна
                       const datetime timel=0, // время
                       const double price1=0, // цена
                       const double time2=0, // время
                       const double price2=0, // цена
                       const double time3=0, // время
                       const double price3=0, // цена
                       const ENUM_ELLIOT_WAVE_DEGREE degree=ELLIOTT_MINUETTE, // степень
                       const bool draw_lines=true, // отображение
                       const color clr=clrRed, // цвет
                       const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль
                       const int width=1, // толщина
                       const bool back=false, // на задний план
                       const bool selection=true, // выделение
                       const bool hidden=true, // скрытие
                       const long z_order=0) // приоритет
{
//--- установим координаты точек привязки, если они не заданы
    ChangeElliotWave3EmptyPoints(timel,price1,time2,price2,time3,price3);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим "Корректирующую волну Эллиота" по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_ELLIOTWAVE3,sub_window,timel,price1,time2,price2,
                     degree,draw_lines,selection,hidden,z_order))
    {
        Print(__FUNCTION__,
              ": не удалось создать \"Корректирующую волну Эллиота\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим степень (размер волны)
    ObjectSetInteger(chart_ID,name,OBJPROP_DEGREE,degree);
//--- включим (true) или отключим (false) режим отображения линий
    ObjectSetInteger(chart_ID,name,OBJPROP_DRAWLINES,draw_lines);
//--- установим цвет объекта
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
}

```

```

//--- установим стиль линий
ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения объекта для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки "Корректирующей волны Эллиота" |
//+-----+
bool ElliotWave3PointChange(const long chart_ID=0,           // ID графика
                           const string name="ElliotWave3", // имя объекта
                           const int point_index=0,        // номер точки привязки
                           datetime time=0,              // координата времени точки
                           double price=0)              // координата цены точки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет "Корректирующую волну Эллиота" |
//+-----+
bool ElliotWave3Delete(const long chart_ID=0,           // ID графика
                      ...

```

```

        const string name="ElliotWave3") // имя объекта
    {
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим объект
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
            ": не удалось удалить \"Корректирующую волну Эллиота\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//----------------------------------------------------------------------------------------------------------------+
//| Проверяет значения точек привязки "Корректирующей волны Эллиота" |
//| и для пустых значений устанавливает значения по умолчанию          |
//----------------------------------------------------------------------------------------------------------------+
void ChangeElliotWave3EmptyPoints(datetime &time1,double &price1,
                                    datetime &time2,double &price2,
                                    datetime &time3,double &price3)
{
//--- массив для приема времени открытия 10 последних баров
    datetime temp[];
    ArrayResize(temp,10);
//--- получим данные
    CopyTime(Symbol(),Period(),TimeCurrent(),10,temp);
//--- получим величину одного пункта на текущем графике
    double point=SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время первой точки не задано, то она будет на 9 баре слева от последнего бара
    if(!time1)
        time1=temp[0];
//--- если цена первой точки не задана, то она будет иметь значение Bid
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она будет на 5 баре слева от последнего бара
    if(!time2)
        time2=temp[4];
//--- если цена второй точки не задана, сдвинем ее на 300 пунктов ниже первой
    if(!price2)
        price2=price1-300*point;
//--- если время третьей точки не задано, то она будет на 1 баре слева от последнего бара
    if(!time3)
        time3=temp[8];
//--- если цена третьей точки не задана, сдвинем ее на 200 пунктов ниже первой
    if(!price3)
        price3=price1-200*point;
}
//----------------------------------------------------------------------------------------------------------------+

```

```

//| Script program start function
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
   InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки объекта
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования "Корректирующей волны Эллиота"
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int d3=InpDate3*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
int p3=InpPrice3*(accuracy-1)/100;
//--- создадим "Корректирующую волну Эллиота"
if(!ElliotWave3Create(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],p
InpDegree,InpDrawLines,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden
{
}

```

```

        return;
    }

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать точки привязки
//--- счетчик цикла
int v_steps=accuracy/5;
//--- перемещаем третью точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p3<accuracy-1)
        p3+=1;
    //--- сдвигаем точку
    if(!ElliotWave3PointChange(0,InpName,2,date[d3],price[p3]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);

//--- счетчик цикла
v_steps=accuracy*4/5;
//--- перемещаем первую и вторую точки привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующие значения
    if(p1>1)
        p1-=1;
    if(p2<accuracy-1)
        p2+=1;
    //--- сдвигаем точки
    if(!ElliotWave3PointChange(0,InpName,0,date[d1],price[p1]))
        return;
    if(!ElliotWave3PointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);

//--- удалим объект с графика

```

```
ElliotWave3Delete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
}
```

## OBJ\_RECTANGLE

Прямоугольник.



### Примечание

Для прямоугольника можно установить режим заливки при помощи свойства [OBJPROP\\_FILL](#).

### Пример

Следующий скрипт создает и перемещает на графике прямоугольник. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит прямоугольник на графике."
#property description "Координаты точек привязки задаются в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="Rectangle"; // Имя прямоугольника
input int         InpDate1=40;          // Дата 1-ой точки в %
input int         InpPrice1=40;         // Цена 1-ой точки в %
input int         InpDate2=60;          // Дата 2-ой точки в %
input int         InpPrice2=60;         // Цена 2-ой точки в %
input color        InpColor=clrRed;       // Цвет прямоугольника
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линий прямоугольника
input int         InpWidth=2;           // Толщина линий прямоугольника

```

```

input bool           InpFill=true;          // Заливка прямоугольника цветом
input bool           InpBack=false;         // Прямоугольника на заднем плане
input bool           InpSelection=true;      // Выделить для перемещений
input bool           InpHidden=true;         // Скрыт в списке объектов
input long           InpZOrder=0;           // Приоритет на нажатие мышью
//+-----+
//| Создает прямоугольник по заданным координатам |+
//+-----+
bool RectangleCreate(const long           chart_ID=0,          // ID графика
                     const string        name="Rectangle", // имя прямоугольника
                     const int            sub_window=0,       // номер подокна
                     datetime            time1=0,           // время первой точки
                     double               price1=0,          // цена первой точки
                     datetime            time2=0,           // время второй точки
                     double               price2=0,          // цена второй точки
                     const color          clr=clrRed,        // цвет прямоугольника
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий прямоугольника
                     const int            width=1,           // толщина линий прямоугольника
                     const bool            fill=false,         // заливка прямоугольника
                     const bool            back=false,        // на заднем плане
                     const bool            selection=true,    // выделить для перемещения
                     const bool            hidden=true,       // скрыт в списке объектов
                     const long            z_order=0)        // приоритет на нажатие
{
//--- установим координаты точек привязки, если они не заданы
    ChangeRectangleEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим прямоугольник по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE,sub_window,time1,price1,time2,price2))
    {
        Print(__FUNCTION__,
              ": не удалось создать прямоугольник! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет прямоугольника
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий прямоугольника
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий прямоугольника
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки прямоугольника
    ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения прямоугольника для перемещения
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки прямоугольника |
//+-----+
bool RectanglePointChange(const long    chart_ID=0,           // ID графика
                          const string name="Rectangle", // имя прямоугольника
                          const int     point_index=0,   // номер точки привязки
                          datetime    time=0,          // координата времени точки
                          double      price=0)        // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет прямоугольник |
//+-----+
bool RectangleDelete(const long    chart_ID=0,           // ID графика
                     const string name="Rectangle") // имя прямоугольника
{
//---бросим значение ошибки
ResetLastError();
//---удалим прямоугольник
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": не удалось удалить прямоугольник! Код ошибки = ",GetLastError());
    return(false);
}

```

```

        }

//--- успешное выполнение
    return(true);
}

//+-----+
//| Проверяет значения точек привязки прямоугольника и для пустых |
//| значений устанавливает значения по умолчанию |
//+-----+

void ChangeRectangleEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
{
    //--- если время первой точки не задано, то она будет на текущем баре
    if(!time1)
        time1=TimeCurrent();

    //--- если цена первой точки не задана, то она будет иметь значение Bid
    if(!price1)
        price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);

    //--- если время второй точки не задано, то она лежит на 9 баров левее второй
    if(!time2)
    {
        //--- массив для приема времени открытия 10 последних баров
        datetime temp[10];
        CopyTime(Symbol(),Period(),time1,10,temp);

        //--- установим вторую точку на 9 баров левее первой
        time2=temp[0];
    }

    //--- если цена второй точки не задана, сдвинем ее на 300 пунктов ниже первой
    if(!price2)
        price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
}

//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
    //--- проверим входные параметры на корректность
    if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
       InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }

    //--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

    //--- размер массива price
    int accuracy=1000;

    //--- массивы для хранения значений дат и цен, которые будут использованы
    //--- для установки и изменения координат точек привязки прямоугольника
    datetime date[];
}

```

```

        double    price[];
//--- выделение памяти
        ArrayResize(date,bars);
        ArrayResize(price,accuracy);
//--- заполним массив дат
        ResetLastError();
        if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
        {
            Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
            return;
        }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
        double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
        double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
        double step=(max_price-min_price)/accuracy;
        for(int i=0;i<accuracy;i++)
            price[i]=min_price+i*step;
//--- определим точки для рисования прямоугольника
        int d1=InpDate1*(bars-1)/100;
        int d2=InpDate2*(bars-1)/100;
        int p1=InpPrice1*(accuracy-1)/100;
        int p2=InpPrice2*(accuracy-1)/100;
//--- создадим прямоугольник
        if(!RectangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
                           InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
        {
            return;
        }
//--- перерисуем график и подождем 1 секунду
        ChartRedraw();
        Sleep(1000);
//--- теперь будем перемещать точки привязки прямоугольника
//--- счетчик цикла
        int h_steps=bars/2;
//--- перемещаем точки привязки
        for(int i=0;i<h_steps;i++)
        {
//--- возьмем следующие значения
            if(d1<bars-1)
                d1+=1;
            if(d2>1)
                d2-=1;
//--- сдвигаем точки
            if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
                return;
            if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
                return;
        }
    }
}

```

```
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}

//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
int v_steps=accuracy/2;
//--- перемещаем точки привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующие значения
    if(p1<accuracy-1)
        p1+=1;
    if(p2>1)
        p2-=1;
    //--- сдвигаем точки
    if(!RectanglePointChange(0,IInpName,0,date[d1],price[p1]))
        return;
    if(!RectanglePointChange(0,IInpName,1,date[d2],price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим прямоугольник с графика
RectangleDelete(0,IInpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_TRIANGLE

Треугольник.



### Примечание

Для треугольника можно установить режим заливки при помощи свойства [OBJPROP\\_FILL](#).

### Пример

Следующий скрипт создает и перемещает на графике треугольник. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит треугольник на графике."
#property description "Координаты точек привязки задаются в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="Triangle";           // Имя треугольника
input int         InpDate1=25;                   // Дата 1-ой точки в %
input int         InpPrice1=50;                  // Цена 1-ой точки в %
input int         InpDate2=70;                   // Дата 2-ой точки в %
input int         InpPrice2=70;                  // Цена 2-ой точки в %
input int         InpDate3=65;                   // Дата 3-ей точки в %
input int         InpPrice3=20;                  // Цена 3-ей точки в %
input color       InpColor=clrRed;               // Цвет треугольника

```

```

input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий треугольника
input int InpWidth=2; // Толщина линий треугольника
input bool InpFill=false; // Заливка треугольника цветом
input bool InpBack=false; // Треугольника на заднем плане
input bool InpSelection=true; // Выделить для перемещений
input bool InpHidden=true; // Скрыт в списке объектов
input long InpZOrder=0; // Приоритет на нажатие мышью
//+-----+
//| Создает треугольник по заданным координатам |
//+-----+
bool TriangleCreate(const long chart_ID=0, // ID графика
                     const string name="Triangle", // имя треугольника
                     const int sub_window=0, // номер подокна
                     datetime timel=0, // время первой точки
                     double price1=0, // цена первой точки
                     datetime time2=0, // время второй точки
                     double price2=0, // цена второй точки
                     datetime time3=0, // время третьей точки
                     double price3=0, // цена третьей точки
                     const color clr=clrRed, // цвет треугольника
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий треугольника
                     const int width=1, // толщина линий треугольника
                     const bool fill=false, // заливка треугольника
                     const bool back=false, // на заднем плане
                     const bool selection=true, // выделить для перемещений
                     const bool hidden=true, // скрыт в списке объектов
                     const long z_order=0) // приоритет на нажатие мышью
{
//--- установим координаты точек привязки, если они не заданы
    ChangeTriangleEmptyPoints(timel,price1,time2,price2,time3,price3);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим треугольник по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_TRIANGLE,sub_window,timel,price1,time2,price2,time3,price3))
    {
        Print(__FUNCTION__,
              ": не удалось создать треугольник! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет треугольника
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий треугольника
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий треугольника
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки треугольника
    ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
}

```

```

//--- включим (true) или отключим (false) режим выделения треугольник для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки треугольника |
//+-----+
bool TrianglePointChange(const long    chart_ID=0,          // ID графика
                         const string name="Triangle", // имя треугольника
                         const int      point_index=0, // номер точки привязки
                         datetime     time=0,        // координата времени точки привязки
                         double       price=0)       // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);

//---бросим значение ошибки
ResetLastError();

//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет треугольник |
//+-----+
bool TriangleDelete(const long    chart_ID=0,          // ID графика
                    const string name="Triangle") // имя треугольника
{
//---бросим значение ошибки
ResetLastError();

//---удалим треугольник
if(!ObjectDelete(chart_ID,name))

```

```

{
    Print(__FUNCTION__,
          ": не удалось удалить треугольник! Код ошибки = ", GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Проверяет значения точек привязки треугольника и для пустых      |
//| значений устанавливает значения по умолчанию                         |
//+-----+
void ChangeTriangleEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2,
                                datetime &time3,double &price3)
{
//--- если время первой точки не задано, то она будет на текущем баре
if(!time1)
    time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
if(!price1)
    price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее второй
if(!time2)
{
//--- массив для приема времени открытия 10 последних баров
datetime temp[10];
CopyTime(Symbol(),Period(),time1,10,temp);
//--- установим вторую точку на 9 баров левее первой
time2=temp[0];
}
//--- если цена второй точки не задана, сдвинем ее на 300 пунктов ниже первой
if(!price2)
    price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время третьей точки не задано, то оно будет совпадать с датой второй точки
if(!time3)
    time3=time2;
//--- если цена первой точки не задана, то она будет совпадать с ценой первой точки
if(!price3)
    price3=price1;
}
//+-----+
//| Script program start function                                         |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
   InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100 ||
   InpDate4<0 || InpDate4>100 || InpPrice4<0 || InpPrice4>100 ||
   InpDate5<0 || InpDate5>100 || InpPrice5<0 || InpPrice5>100 ||
   InpDate6<0 || InpDate6>100 || InpPrice6<0 || InpPrice6>100 ||
   InpDate7<0 || InpDate7>100 || InpPrice7<0 || InpPrice7>100 ||
   InpDate8<0 || InpDate8>100 || InpPrice8<0 || InpPrice8>100 ||
   InpDate9<0 || InpDate9>100 || InpPrice9<0 || InpPrice9>100 ||
   InpDate10<0 || InpDate10>100 || InpPrice10<0 || InpPrice10>100)
}

```

```

InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки треугольника
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования треугольника
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int d3=InpDate3*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
int p3=InpPrice3*(accuracy-1)/100;
//--- создадим треугольник
if(!TriangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
    InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точки привязки треугольника
//--- счетчик цикла

```

```

int v_steps=accuracy*3/10;
//--- перемещаем первую точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p1>1)
        p1-=1;
    //--- сдвигаем точку
    if(!TrianglePointChange(0,InpName,0,date[d1],price[p1]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
int h_steps=bars*9/20-1;
//--- перемещаем вторую точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d2>1)
        d2-=1;
    //--- сдвигаем точку
    if(!TrianglePointChange(0,InpName,1,date[d2],price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
v_steps=accuracy/4;
//--- перемещаем третью точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p3<accuracy-1)
        p3+=1;
    //--- сдвигаем точку
    if(!TrianglePointChange(0,InpName,2,date[d3],price[p3]))

```

```
    return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);
//--- удалим треугольник с графика
TriangleDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
}
```

## OBJ\_ELLIPSE

Эллипс.



### Примечание

Для эллипса можно установить режим заливки при помощи свойства [OBJPROP\\_FILL](#).

### Пример

Следующий скрипт создает и перемещает на графике эллипс. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
---- описание
#property description "Скрипт строит эллипс на графике."
#property description "Координаты точек привязки задаются"
#property description "в процентах от размеров окна графика."
---- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
---- входные параметры скрипта
input string      InpName="Ellipse";           // Имя эллипса
input int         InpDate1=30;                  // Дата 1-ой точки в %
input int         InpPrice1=20;                 // Цена 1-ой точки в %
input int         InpDate2=70;                  // Дата 2-ой точки в %
input int         InpPrice2=80;                 // Цена 2-ой точки в %
input int         InpDate3=50;                  // Дата 3-ей точки в %
input int         InpPrice3=60;                 // Цена 3-ей точки в %
input color        InpColor=clrRed;             // Цвет эллипса
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий эллипса
```

```

input int          InpWidth=2;           // Толщина линий эллипса
input bool         InpFill=false;        // Заливка эллипса цветом
input bool         InpBack=false;        // Эллипс на заднем плане
input bool         InpSelection=true;     // Выделить для перемещений
input bool         InpHidden=true;       // Скрыт в списке объектов
input long         InpZOrder=0;          // Приоритет на нажатие мышью
//+-----+
//| Создает эллипс по заданным координатам |
//+-----+
bool EllipseCreate(const long          chart_ID=0,          // ID графика
                    const string        name="Ellipse",      // имя эллипса
                    const int           sub_window=0,       // номер подокна
                    const datetime       time1=0,          // время первой точки
                    const double         price1=0,          // цена первой точки
                    const datetime       time2=0,          // время второй точки
                    const double         price2=0,          // цена второй точки
                    const datetime       time3=0,          // время третьей точки
                    const double         price3=0,          // цена третьей точки
                    const color          clr=clrRed,        // цвет эллипса
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий эллипса
                    const int            width=1,          // толщина линий эллипса
                    const bool           fill=false,        // заливка эллипса цветом
                    const bool           back=false,        // на заднем плане
                    const bool           selection=true,    // выделить для перемещений
                    const bool           hidden=true,       // скрыт в списке объектов
                    const long           z_order=0)        // приоритет на нажатие мышью
{
//--- установим координаты точек привязки, если они не заданы
    ChangeEllipseEmptyPoints(time1,price1,time2,price2,time3,price3);
//---бросим значение ошибки
    ResetLastError();
//--- создадим эллипс по заданным координатам
    if(!ObjectCreate(chart_ID,name,OBJ_ELLIPSE,sub_window,time1,price1,time2,price2,tir
    {
        Print(__FUNCTION__,
              ": не удалось создать эллипс! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет эллипса
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий эллипса
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий эллипса
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки эллипса
    ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения эллипса для перемещений
}

```

```

//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки эллипса |
//+-----+
bool EllipsePointChange(const long    chart_ID=0,      // ID графика
                        const string name="Ellipse", // имя эллипса
                        const int     point_index=0, // номер точки привязки
                        datetime    time=0,        // координата времени точки привязки
                        double       price=0)     // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,point_index,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет эллипс |
//+-----+
bool EllipseDelete(const long    chart_ID=0,      // ID графика
                   const string name="Ellipse") // имя эллипса
{
//---бросим значение ошибки
ResetLastError();
//---удалим эллипс
if(!ObjectDelete(chart_ID,name))
{
}

```

```

Print(__FUNCTION__,
      ": не удалось удалить эллипс! Код ошибки = ", GetLastError());
return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Проверяет значения точек привязки эллипса и для пустых значений |
//| устанавливает значения по умолчанию                                |
//+-----+
void ChangeEllipseEmptyPoints(datetime &time1,double &price1,
                               datetime &time2,double &price2,
                               datetime &time3,double &price3)
{
//--- если время первой точки не задано, то она будет на текущем баре
if(!time1)
    time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
if(!price1)
    price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее второй
if(!time2)
{
//--- массив для приема времени открытия 10 последних баров
datetime temp[10];
CopyTime(Symbol(),Period(),time1,10,temp);
//--- установим вторую точку на 9 баров левее первой
time2=temp[0];
}
//--- если цена второй точки не задана, сдвинем ее на 300 пунктов ниже первой
if(!price2)
    price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время третьей точки не задано, то оно будет совпадать с датой второй точки
if(!time3)
    time3=time2;
//--- если цена третьей точки не задана, то она будет совпадать с ценой первой точки
if(!price3)
    price3=price1;
}

//+-----+
//| Script program start function                                         |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
   InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
   InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
}

```

```

{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}

//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

//--- размер массива price
int accuracy=1000;

//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки эллипса
datetime date[];
double price[];

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования эллипса
int d1=InpDate1*(bars-1)/100;
int d2=InpDate2*(bars-1)/100;
int d3=InpDate3*(bars-1)/100;
int p1=InpPrice1*(accuracy-1)/100;
int p2=InpPrice2*(accuracy-1)/100;
int p3=InpPrice3*(accuracy-1)/100;

//--- создадим эллипс
if(!EllipseCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
    InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать точки привязки эллипса
//--- счетчик цикла
int v_steps=accuracy/5;

```

```

//--- перемещаем первую и вторую точки привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующие значения
    if(p1<accuracy-1)
        p1+=1;
    if(p2>1)
        p2-=1;
    //--- сдвигаем точки
    if(!EllipsePointChange(0,IvpName,0,date[d1],price[p1]))
        return;
    if(!EllipsePointChange(0,IvpName,1,date[d2],price[p2]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);

//--- счетчик цикла
int h_steps=bars/5;
//--- перемещаем третью точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d3>1)
        d3-=1;
    //--- сдвигаем точку
    if(!EllipsePointChange(0,IvpName,2,date[d3],price[p3]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}

//--- задержка в 1 секунду
Sleep(1000);

//--- удалим эллипс с графика
EllipseDelete(0,IvpName);
ChartRedraw();

//--- задержка в 1 секунду
Sleep(1000);

//---
}

```

## OBJ\_ARROW\_THUMB\_UP

Знак "Хорошо".



### Примечание

Положение точки привязки относительно знака можно выбрать из перечисления [ENUM\\_ARROW\\_ANCHOR](#).

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства [OBJPROP\\_WIDTH](#) при написании кода в MetaEditor.

### Пример

Следующий скрипт создает и перемещает на графике знак "Хорошо". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт рисует знак \"Хорошо\" (большой палец вверх)."
#property description "Координата точки привязки задается в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="ThumbUp";           // Имя знака
input int       InpDate=75;                     // Дата точки привязки в %
input int       InpPrice=25;                    // Цена точки привязки в %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Способ привязки
input color      InpColor=clrRed;               // Цвет знака

```

```

input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;      // Стиль окаймляющей линии
input int                InpWidth=5;            // Размер знака
input bool               InpBack=false;         // Знак на заднем плане
input bool               InpSelection=true;       // Выделить для перемещений
input bool               InpHidden=true;          // Скрыт в списке объектов
input long               InpZOrder=0;           // Приоритет на нажатие мышью
//+-----+
//| Создает знак "Хорошо" |
//+-----+
bool ArrowThumbUpCreate(const long             chart_ID=0,           // ID графика
                        const string          name="ThumbUp",        // имя знака
                        const int              sub_window=0,        // номер подокна
                        datetime              time=0,            // время точки привязки
                        double                price=0,           // цена точки привязки
                        const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привязки
                        const color             clr=clrRed,          // цвет знака
                        const ENUM_LINE_STYLE  style=STYLE_SOLID,     // стиль окаймляющей линии
                        const int              width=3,            // размер знака
                        const bool             back=false,          // на заднем плане
                        const bool             selection=true,       // выделить для перемещения
                        const bool             hidden=true,          // скрыт в списке
                        const long             z_order=0)           // приоритет на переднем плане
{
//--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим знак
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_UP,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать знак \"Хорошо\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим способ привязки
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
}

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
// | Перемещает точку привязки |
//+-----+
bool ArrowThumbUpMove(const long    chart_ID=0,      // ID графика
                      const string name="ThumbUp", // имя объекта
                      datetime     time=0,        // координата времени точки привязки
                      double       price=0)      // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
// | Меняет способ привязки знака "Хорошо" |
//+-----+
bool ArrowThumbUpAnchorChange(const long    chart_ID=0,      // ID графика
                             const string name="ThumbUp", // имя объекта
                             const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ привязки
{
//---бросим значение ошибки
ResetLastError();
//---изменим способ привязки
if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
{
    Print(__FUNCTION__,
          ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
    return(false);
}
}

```

```

//--- успешное выполнение
    return(true);
}
//+-----+
//| Удаляет знак "Хорошо" |
//+-----+
bool ArrowThumbUpDelete(const long    chart_ID=0,          // ID графика
                        const string name="ThumbUp") // имя знака
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим знак
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить знак \"Хорошо\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Проверяет значения точки привязки и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- если время точки не задано, то она будет на текущем баре
    if(!time)
        time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }
//--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
    int accuracy=1000;
}

```

```

//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
datetime date[];
double price[];

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования знака
int d=InpDate*(bars-1)/100;
int p=InpPrice*(accuracy-1)/100;

//--- создадим знак "Хорошо" на графике
if(!ArrowThumbUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
int h_steps=bars/4;
//--- перемещаем точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d>1)
        d-=1;
    //--- сдвигаем точку
    if(!ArrowThumbUpMove(0,InpName,date[d],price[p]))
        return;
}

//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;

```

```
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}

//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
int v_steps=accuracy/4;
//--- перемещаем точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p<accuracy-1)
        p+=1;
    //--- сдвигаем точку
    if(!ArrowThumbUpMove(0,IInpName,date[d],price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}
//--- меняем положение точки привязки относительно знака
ArrowThumbUpAnchorChange(0,IInpName,ANCHOR_BOTTOM);
//--- перерисуем график
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим знак с графика
ArrowThumbUpDelete(0,IInpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_ARROW\_THUMB\_DOWN

Знак "Плохо".



### Примечание

Положение точки привязки относительно знака можно выбрать из перечисления [ENUM\\_ARROW\\_ANCHOR](#).

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства [OBJPROP\\_WIDTH](#) при написании кода в MetaEditor.

### Пример

Следующий скрипт создает и перемещает на графике знак "Плохо". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт рисует знак \"Плохо\" (большой палец вниз)."
#property description "Координата точки привязки задается в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="ThumbDown";      // Имя знака
input int                InpDate=25;              // Дата точки привязки в %
input int                InpPrice=75;             // Цена точки привязки в %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Способ привязки

```

```

input color           InpColor=clrRed;           // Цвет знака
input ENUM_LINE_STYLE InpStyle=STYLE_DOT;       // Стиль окаймляющей линии
input int             InpWidth=5;              // Размер знака
input bool            InpBack=false;           // Знак на заднем плане
input bool            InpSelection=true;         // Выделить для перемещений
input bool            InpHidden=true;           // Скрыт в списке объектов
input long            InpZOrder=0;              // Приоритет на нажатие мышью
//+-----+
//| Создает знак "Плохо"
//+-----+
bool ArrowThumbDownCreate(const long           chart_ID=0,          // ID графика
                           const string        name="ThumbDown",   // имя знака
                           const int           sub_window=0,     // номер подокна
                           const datetime      time=0,          // время точки
                           const double         price=0,         // цена точки
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привязки
                           const color          clr=clrRed,       // цвет знака
                           const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль окаймляющей линии
                           const int           width=3,          // размер знака
                           const bool           back=false,        // на заднем плане
                           const bool           selection=true,    // выделить для перемещения
                           const bool           hidden=true,      // скрыт в списке объектов
                           const long           z_order=0)        // приоритет на нажатие мышью
{
//--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим знак
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_DOWN,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать знак \"Плохо\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим способ привязки
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки |
//+-----+
bool ArrowThumbDownMove(const long    chart_ID=0,           // ID графика
                        const string name="ThumbDown", // имя объекта
                        datetime   time=0,            // координата времени точки привязки
                        double     price=0)          // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Меняет способ привязки знака "Плохо" |
//+-----+
bool ArrowThumbDownAnchorChange(const long    chart_ID=0,           // ID графика
                                 const string name="ThumbDown", // имя объекта
                                 const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ привязки
{
//---бросим значение ошибки
ResetLastError();
//---изменим способ привязки
if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
{
    Print(__FUNCTION__,
          ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
    return(false);
}

```

```

        }

//--- успешное выполнение
    return(true);
}

//+-----+
//| Удаляет знак "Плохо" |
//+-----+
bool ArrowThumbDownDelete(const long    chart_ID=0,           // ID графика
                           const string name="ThumbDown") // имя знака
{
    //--- сбросим значение ошибки
    ResetLastError();

    //--- удалим знак
    if(!ObjectDelete(chart_ID, name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить знак \"Плохо\"! Код ошибки = ", GetLastError());
        return(false);
    }

    //--- успешное выполнение
    return(true);
}

//+-----+
//| Проверяет значения точки привязки и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
    //--- если время точки не задано, то она будет на текущем баре
    if(!time)
        time=TimeCurrent();

    //--- если цена точки не задана, то она будет иметь значение Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);

}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    //--- проверим входные параметры на корректность
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }

    //--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

    //--- размер массива price
}

```

```

int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования знака
int d=InpDate*(bars-1)/100;
int p=InpPrice*(accuracy-1)/100;
//--- создадим знак "Плохо" на графике
if(!ArrowThumbDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
int h_steps=bars/4;
//--- перемещаем точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d<bars-1)
        d+=1;
    //--- сдвигаем точку
    if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())

```

```
    return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}

//--- задержка в 1 секунду
Sleep(1000);
//--- счетчик цикла
int v_steps=accuracy/4;
//--- перемещаем точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p>1)
        p-=1;
    //--- сдвигаем точку
    if(!ArrowThumbDownMove(0,IInpName,date[d],price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}

//--- меняем положение точки привязки относительно знака
ArrowThumbDownAnchorChange(0,IInpName,ANCHOR_TOP);
//--- перерисуем график
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим знак с графика
ArrowThumbDownDelete(0,IInpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_ARROW\_UP

Знак "Стрелка вверх".



### Примечание

Положение точки привязки относительно знака можно выбрать из перечисления [ENUM\\_ARROW\\_ANCHOR](#).

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства [OBJPROP\\_WIDTH](#) при написании кода в MetaEditor.

### Пример

Следующий скрипт создает и перемещает на графике знак "Стрелка вверх". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт рисует знак \"Стрелка вверх\"."
#property description "Координата точки привязки задается в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="ArrowUp";           // Имя знака
input int       InpDate=25;                     // Дата точки привязки в %
input int       InpPrice=25;                    // Цена точки привязки в %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Способ привязки
input color      InpColor=clrRed;               // Цвет знака

```

```

input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;      // Стиль окаймляющей линии
input int                InpWidth=5;            // Размер знака
input bool               InpBack=false;         // Знак на заднем плане
input bool               InpSelection=false;    // Выделить для перемещений
input bool               InpHidden=true;        // Скрыт в списке объектов
input long               InpZOrder=0;          // Приоритет на нажатие мышью
//+-----+
//| Создает знак "Стрелка вверх"
//+-----+
bool ArrowUpCreate(const long           chart_ID=0,           // ID графика
                    const string        name="ArrowUp",        // имя знака
                    const int            sub_window=0,       // номер подокна
                    const datetime       time=0,             // время точки привязки
                    const double          price=0,            // цена точки привязки
                    const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привязки
                    const color           clr=clrRed,          // цвет знака
                    const ENUM_LINE_STYLE style=STYLE_SOLID,   // стиль окаймляющей
                    const int              width=3,            // размер знака
                    const bool             back=false,         // на заднем плане
                    const bool             selection=true,     // выделить для перемещения
                    const bool             hidden=true,        // скрыт в списке объектов
                    const long             z_order=0)          // приоритет на нажатие
{
//--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим знак
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_UP,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать знак \"Стрелка вверх\"! Код ошибки = ",GetLastError);
        return(false);
    }
//--- установим способ привязки
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки |
//+-----+
bool ArrowUpMove(const long    chart_ID=0,           // ID графика
                  const string name="ArrowUp", // имя объекта
                  datetime     time=0,        // координата времени точки привязки
                  double       price=0)      // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Меняет способ привязки знака "Стрелка вверх" |
//+-----+
bool ArrowUpAnchorChange(const long            chart_ID=0,           // ID графика
                         const string         name="ArrowUp", // имя объекта
                         const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ привязки
{
//---бросим значение ошибки
ResetLastError();
//---изменим положение точки привязки
if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
{
    Print(__FUNCTION__,
          ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
    return(false);
}
}

```

```

//--- успешное выполнение
    return(true);
}
//+-----+
//| Удаляет знак "Стрелка вверх" |
//+-----+
bool ArrowUpDelete(const long    chart_ID=0,          // ID графика
                    const string name="ArrowUp") // имя знака
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим знак
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить знак \"Стрелка вверх\"! Код ошибки = ",GetLastError
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Проверяет значения точки привязки и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- если время точки не задано, то она будет на текущем баре
    if(!time)
        time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }
//--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
    int accuracy=1000;
}

```

```

//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
datetime date[];
double price[];

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования знака
int d=InpDate*(bars-1)/100;
int p=InpPrice*(accuracy-1)/100;

//--- создадим знак "Стрелка вверх" на графике
if(!ArrowUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
int v_steps=accuracy/2;
//--- перемещаем точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p<accuracy-1)
        p+=1;
    //--- сдвигаем точку
    if(!ArrowUpMove(0,InpName,date[d],price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
}

```

```
//--- перерисуем график
ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);

//--- меняем положение точки привязки относительно знака
ArrowUpAnchorChange(0, InpName, ANCHOR_BOTTOM);

//--- перерисуем график
ChartRedraw();

//--- задержка в 1 секунду
Sleep(1000);

//--- удалим знак с графика
ArrowUpDelete(0, InpName);

ChartRedraw();

//--- задержка в 1 секунду
Sleep(1000);

//---
```

## OBJ\_ARROW\_DOWN

Знак "Стрелка вниз".



### Примечание

Положение точки привязки относительно знака можно выбрать из перечисления [ENUM\\_ARROW\\_ANCHOR](#).

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства [OBJPROP\\_WIDTH](#) при написании кода в MetaEditor.

### Пример

Следующий скрипт создает и перемещает на графике знак "Стрелка вниз". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует знак \"Стрелка вниз\"."
#property description "Координата точки привязки задается в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="ArrowDown";          // Имя знака
input int                InpDate=75;                  // Дата точки привязки в %
input int                InpPrice=75;                 // Цена точки привязки в %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM;    // Способ привязки
input color              InpColor=clrRed;             // Цвет знака
input ENUM_LINE_STYLE    InpStyle=STYLE_DOT;          // Стиль окаймляющей линии
```

```

input int           InpWidth=5;           // Размер знака
input bool          InpBack=false;        // Знак на заднем плане
input bool          InpSelection=false;    // Выделить для перемещений
input bool          InpHidden=true;        // Скрыт в списке объектов
input long          InpZOrder=0;          // Приоритет на нажатие мышью
//+-----+
//| Создает знак "Стрелка вниз" |
//+-----+
bool ArrowDownCreate(const long           chart_ID=0,           // ID графика
                      const string         name="ArrowDown",      // имя знака
                      const int            sub_window=0,       // номер подокна
                      datetime            time=0,             // время точки привязки
                      double               price=0,            // цена точки привязки
                      const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привязки
                      const color           clr=clrRed,          // цвет знака
                      const ENUM_LINE_STYLE style=STYLE_SOLID,   // стиль окаймляющей линии
                      const int            width=3,            // размер знака
                      const bool            back=false,          // на заднем плане
                      const bool            selection=true,       // выделить для перемещения
                      const bool            hidden=true,          // скрыт в списке объектов
                      const long            z_order=0)          // приоритет на нажатие
{
//--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим знак
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_DOWN,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать знак \"Стрелка вниз\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- способ привязки
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

```

```

//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки |
//+-----+
bool ArrowDownMove(const long    chart_ID=0,           // ID графика
                    const string   name="ArrowDown", // имя объекта
                    datetime      time=0,          // координата времени точки привязки
                    double        price=0)        // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Меняет способ привязки знака "Стрелка вниз" |
//+-----+
bool ArrowDownAnchorChange(const long            chart_ID=0,           // ID графика
                           const string          name="ArrowDown", // имя объекта
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ привязки
{
//---бросим значение ошибки
ResetLastError();
//---изменим положение точки привязки
if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
{
    Print(__FUNCTION__,
          ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение

```

```

        return(true);
    }
//+-----+
//| Удаляет знак "Стрелка вниз" |
//+-----+
bool ArrowDownDelete(const long chart_ID=0,           // ID графика
                     const string name="ArrowDown") // имя знака
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим знак
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить знак \"Стрелка вниз\"! Код ошибки = ", GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Проверяет значения точки привязки и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- если время точки не задано, то она будет на текущем баре
    if(!time)
        time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }
//--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
    int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
}

```

```

//--- для установки и изменения координат точки привязки знака
datetime date[];
double price[];

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования знака
int d=InpDate*(bars-1)/100;
int p=InpPrice*(accuracy-1)/100;

//--- создадим знак "Стрелка вниз" на графике
if(!ArrowDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
int v_steps=accuracy/2;

//--- перемещаем точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p>1)
        p-=1;
    //--- сдвигаем точку
    if(!ArrowDownMove(0,InpName,date[d],price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
}

//--- перерисуем график

```

```
    ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);

//--- меняем положение точки привязки относительно знака
ArrowDownAnchorChange(0, InpName, ANCHOR_TOP);

//--- перерисуем график
ChartRedraw();

//--- задержка в 1 секунду
Sleep(1000);

//--- удалим знак с графика
ArrowDownDelete(0, InpName);

ChartRedraw();

//--- задержка в 1 секунду
Sleep(1000);

//---
```

## OBJ\_ARROW\_STOP

Знак "Стоп".



### Примечание

Положение точки привязки относительно знака можно выбрать из перечисления [ENUM\\_ARROW\\_ANCHOR](#).

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства [OBJPROP\\_WIDTH](#) при написании кода в MetaEditor.

### Пример

Следующий скрипт создает и перемещает на графике знак "Стоп". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт рисует знак \"Стоп\"."
#property description "Координата точки привязки задается в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="ArrowStop";      // Имя знака
input int              InpDate=10;                // Дата точки привязки в %
input int              InpPrice=50;               // Цена точки привязки в %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Способ привязки
input color             InpColor=clrRed;          // Цвет знака

```

```

input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;           // Стиль окаймляющей линии
input int                InpWidth=5;                  // Размер знака
input bool               InpBack=false;              // Знак на заднем плане
input bool               InpSelection=false;          // Выделить для перемещений
input bool               InpHidden=true;              // Скрыт в списке объектов
input long               InpZOrder=0;                 // Приоритет на нажатие мышью
//+-----+
//| Создает знак "Стоп"
//+-----+
bool ArrowStopCreate(const long             chart_ID=0,           // ID графика
                      const string          name="ArrowStop",      // имя знака
                      const int              sub_window=0,        // номер подокна
                      datetime              time=0,            // время точки привязки
                      double                price=0,           // цена точки привязки
                      const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привязки
                      const color             clr=clrRed,         // цвет знака
                      const ENUM_LINE_STYLE  style=STYLE_SOLID,     // стиль окаймляющей линии
                      const int              width=3,            // размер знака
                      const bool              back=false,          // на заднем плане
                      const bool              selection=true,       // выделить для перемещения
                      const bool              hidden=true,         // скрыт в списке объектов
                      const long              z_order=0)           // приоритет на нажатие
{
    //--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);
    //--- сбросим значение ошибки
    ResetLastError();
    //--- создадим знак
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_STOP,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать знак \"Стоп\"! Код ошибки = ",GetLastError());
        return(false);
    }
    //--- установим способ привязки
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
    //--- установим цвет знака
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- установим стиль окаймляющей линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- установим размер знака
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- включим (true) или отключим (false) режим перемещения знака мышью
    //--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
    //--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
    //--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
}

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
// | Перемещает точку привязки
//+-----+
bool ArrowStopMove(const long    chart_ID=0,           // ID графика
                    const string name="ArrowStop", // имя объекта
                    datetime      time=0,        // координата времени точки привязки
                    double        price=0)       // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
// | Меняет способ привязки знака "Стоп"
//+-----+
bool ArrowStopAnchorChange(const long    chart_ID=0,           // ID графика
                           const string name="ArrowStop", // имя объекта
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // положение точки привязки
{
//---бросим значение ошибки
ResetLastError();
//---изменим способ привязки
if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
{
    Print(__FUNCTION__,
          ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
    return(false);
}
}

```

```

//--- успешное выполнение
    return(true);
}
//+-----+
//| Удаляет знак "Стоп" |
//+-----+
bool ArrowStopDelete(const long chart_ID=0,           // ID графика
                      const string name="ArrowStop") // имя знака
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим знак
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить знак \"Стоп\"! Код ошибки = ", GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Проверяет значения точки привязки и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- если время точки не задано, то она будет на текущем баре
    if(!time)
        time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }
//--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
    int accuracy=1000;
}

```

```

//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
datetime date[];
double price[];

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования знака
int d=InpDate*(bars-1)/100;
int p=InpPrice*(accuracy-1)/100;

//--- создадим знак "Стоп" на графике
if(!ArrowStopCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
int h_steps=bars*2/5;
//--- перемещаем точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d<bars-1)
        d+=1;
    //--- сдвигаем точку
    if(!ArrowStopMove(0,InpName,date[d],price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
}

```

```
//--- перерисуем график
ChartRedraw();
// задержка в 0.025 секунды
Sleep(25);
}

//--- меняем положение точки привязки относительно знака
ArrowStopAnchorChange(0, InpName, ANCHOR_TOP);
//--- перерисуем график
ChartRedraw();
//--- счетчик цикла
h_steps=bars*2/5;
//--- перемещаем точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d<bars-1)
        d+=1;
    //--- сдвигаем точку
    if(!ArrowStopMove(0, InpName, date[d], price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.025 секунды
    Sleep(25);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим знак с графика
ArrowStopDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_ARROW\_CHECK

Знак "Галка".



### Примечание

Положение точки привязки относительно знака можно выбрать из перечисления [ENUM\\_ARROW\\_ANCHOR](#).

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства [OBJPROP\\_WIDTH](#) при написании кода в MetaEditor.

### Пример

Следующий скрипт создает и перемещает на графике знак "Галка". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт рисует знак \"Галка\"."
#property description "Координата точки привязки задается в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="ArrowCheck"; // Имя знака
input int             InpDate=10;           // Дата точки привязки в %
input int             InpPrice=50;          // Цена точки привязки в %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Способ привязки
input color            InpColor=clrRed;        // Цвет знака

```

```

input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;      // Стиль окаймляющей линии
input int                InpWidth=5;            // Размер знака
input bool               InpBack=false;         // Знак на заднем плане
input bool               InpSelection=false;    // Выделить для перемещений
input bool               InpHidden=true;        // Скрыт в списке объектов
input long               InpZOrder=0;          // Приоритет на нажатие мышью
//+-----+
//| Создает знак "Галка"
//+-----+
bool ArrowCheckCreate(const long           chart_ID=0,           // ID графика
                      const string        name="ArrowCheck", // имя знака
                      const int            sub_window=0,       // номер подокна
                      const datetime       time=0,           // время точки привязки
                      const double          price=0,          // цена точки привязки
                      const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привязки
                      const color           clr=clrRed,        // цвет знака
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль окаймляющей линии
                      const int             width=3,           // размер знака
                      const bool            back=false,        // на заднем плане
                      const bool            selection=true,    // выделить для перемещения
                      const bool            hidden=true,       // скрыт в списке
                      const long            z_order=0)        // приоритет на нажатие
{
//--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим знак
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_CHECK,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать знак \"Галка\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим способ привязки
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
}

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
// | Перемещает точку привязки
//+-----+
bool ArrowCheckMove(const long chart_ID=0,           // ID графика
                     const string name="ArrowCheck", // имя объекта
                     datetime    time=0,          // координата времени точки привязки
                     double      price=0)         // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
// | Меняет способ привязки "Галки"
//+-----+
bool ArrowCheckAnchorChange(const long             chart_ID=0,           // ID графика
                            const string          name="ArrowCheck", // имя объекта
                            const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ привязки
{
//---бросим значение ошибки
ResetLastError();
//---изменим способ привязки
if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
{
    Print(__FUNCTION__,
          ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
    return(false);
}
}

```

```

//--- успешное выполнение
    return(true);
}
//+-----+
//| Удаляет знак "Галка"
//+-----+
bool ArrowCheckDelete(const long    chart_ID=0,           // ID графика
                      const string name="ArrowCheck") // имя знака
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим знак
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить знак \"Галка\"! Код ошибки = ", GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Проверяет значения точки привязки и для пустых значений
//| устанавливает значения по умолчанию
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- если время точки не задано, то она будет на текущем баре
    if(!time)
        time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
    if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
    {
        Print("Ошибка! Некорректные значения входных параметров!");
        return;
    }
//--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
    int accuracy=1000;
}

```

```

//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
datetime date[];
double price[];

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования знака
int d=InpDate*(bars-1)/100;
int p=InpPrice*(accuracy-1)/100;

//--- создадим знак "Галка" на графике
if(!ArrowCheckCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
int h_steps=bars*2/5;
//--- перемещаем точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d<bars-1)
        d+=1;
    //--- сдвигаем точку
    if(!ArrowCheckMove(0,InpName,date[d],price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
}

```

```
//--- перерисуем график
ChartRedraw();
// задержка в 0.025 секунды
Sleep(25);
}

//--- меняем положение точки привязки относительно знака
ArrowCheckAnchorChange(0, InpName, ANCHOR_BOTTOM);
//--- перерисуем график
ChartRedraw();
//--- счетчик цикла
h_steps=bars*2/5;
//--- перемещаем точку привязки
for(int i=0;i<h_steps;i++)
{
    //--- возьмем следующее значение
    if(d<bars-1)
        d+=1;
    //--- сдвигаем точку
    if(!ArrowCheckMove(0, InpName, date[d], price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.025 секунды
    Sleep(25);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим знак с графика
ArrowCheckDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_ARROW\_LEFT\_PRICE

Левая ценовая метка.



### Пример

Следующий скрипт создает и перемещает на графике левую ценовую метку. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает левую ценовую метку на графике."
#property description "Координата точки привязки задается в процентах"
#property description "от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="LeftPrice"; // Имя ценовой метки
input int              InpDate=100;        // Дата точки привязки в %
input int              InpPrice=10;        // Цена точки привязки в %
input color            InpColor=clrRed;    // Цвет ценовой метки
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID; // Стиль окаймляющей линии
input int              InpWidth=2;         // Размер ценовой метки
input bool             InpBack=false;      // Метка на заднем плане
input bool             InpSelection=true;   // Выделить для перемещений
input bool             InpHidden=true;     // Скрыт в списке объектов
input long             InpZOrder=0;        // Приоритет на нажатие мышью
//+-----+
//| Создает левую ценовую метку |
```

```

//+-----+
bool ArrowLeftPriceCreate(const long           chart_ID=0,          // ID графика
                           const string        name="LeftPrice",   // имя ценовой метки
                           const int            sub_window=0,      // номер подокна
                           const datetime       time=0,           // время точки привязки
                           const double          price=0,          // цена точки привязки
                           const color           clr=clrRed,        // цвет ценовой метки
                           const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль окаймляющей линии
                           const int             width=1,          // размер ценовой метки
                           const bool            back=false,        // на заднем плане
                           const bool            selection=true,    // выделить для перевода в список
                           const bool            hidden=true,       // скрыт в списке
                           const long            z_order=0)        // приоритет на экране

{
//--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим ценовую метку
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_LEFT_PRICE,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать левую ценовую метку! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет метки
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер метки
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
    return(true);
}
//+-----+
//| Перемещает точку привязки
//+-----+

```

```

bool ArrowLeftPriceMove(const long    chart_ID=0,           // ID графика
                        const string name="LeftPrice", // имя метки
                        datetime     time=0,          // координата времени точки привязки
                        double       price=0)        // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//---переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет левую ценовую метку с графика |
//+-----+
bool ArrowLeftPriceDelete(const long    chart_ID=0,           // ID графика
                          const string name="LeftPrice") // имя метки
{
//---бросим значение ошибки
ResetLastError();
//---удалим метку
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": не удалось удалить левую ценовую метку! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Проверяет значения точки привязки и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- если время точки не задано, то она будет на текущем баре
if(!time)
    time=TimeCurrent();
}

```

```

//--- если цена точки не задана, то она будет иметь значение Bid
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//+-----+
//| Script program start function | 
//+-----+

void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}

//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

//--- размер массива price
int accuracy=1000;

//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки метки
datetime date[];
double price[];

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования метки
int d=InpDate*(bars-1)/100;
int p=InpPrice*(accuracy-1)/100;

//--- создадим левую ценовую метку на графике
if(!ArrowLeftPriceCreate(0,InpName,0,date[d],price[p],InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}

```

```
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать точку привязки
//--- счетчик цикла
int v_steps=accuracy*4/5;
//--- перемещаем точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p<accuracy-1)
        p+=1;
    //--- сдвигаем точку
    if(!ArrowLeftPriceMove(0,IopName,date[d],price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);

//--- удалим метку с графика
ArrowLeftPriceDelete(0,IopName);
ChartRedraw();

//--- задержка в 1 секунду
Sleep(1000);

//---
```

## OBJ\_ARROW\_RIGHT\_PRICE

Правая ценовая метка.



### Пример

Следующий скрипт создает и перемещает на графике правую ценовую метку. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает правую ценовую метку на графике."
#property description "Координата точки привязки задается в процентах"
#property description "от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="RightPrice"; // Имя ценовой метки
input int              InpDate=0;           // Дата точки привязки в %
input int              InpPrice=90;          // Цена точки привязки в %
input color            InpColor=clrRed;       // Цвет ценовой метки
input ENUM_LINE_STYLE  InpStyle=STYLE_SOLID; // Стиль окаймляющей линии
input int              InpWidth=2;          // Размер ценовой метки
input bool             InpBack=false;        // Метка на заднем плане
input bool             InpSelection=true;     // Выделить для перемещений
input bool             InpHidden=true;       // Скрыт в списке объектов
input long             InpZOrder=0;          // Приоритет на нажатие мышью
//+-----+
//| Создает правую ценовую метку |
```

```

//+-----+
bool ArrowRightPriceCreate(const long           chart_ID=0,          // ID графика
                           const string        name="RightPrice", // имя ценовой метки
                           const int            sub_window=0,      // номер подокна
                           const datetime       time=0,           // время точки привязки
                           const double          price=0,          // цена точки привязки
                           const color           clr=clrRed,        // цвет ценовой метки
                           const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль окаймляющей линии
                           const int             width=1,          // размер ценовой метки
                           const bool            back=false,        // на заднем плане
                           const bool            selection=true,    // выделить для перетаскивания
                           const bool            hidden=true,       // скрыт в списке объектов
                           const long            z_order=0)        // приоритет на получение события нажатия мыши
{
//--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим ценовую метку
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_RIGHT_PRICE,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать правую ценовую метку! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим цвет метки
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер метки
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
    return(true);
}
//+-----+
//| Перемещает точку привязки
//+-----+

```

```

bool ArrowRightPriceMove(const long    chart_ID=0,           // ID графика
                         const string name="RightPrice", // имя метки
                         datetime   time=0,            // координата времени точки
                         double     price=0)          // координата цены точки при
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//---переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет правую ценовую метку с графика |
//+-----+
bool ArrowRightPriceDelete(const long    chart_ID=0,           // ID графика
                           const string name="RightPrice") // имя метки
{
//---бросим значение ошибки
ResetLastError();
//---удалим метку
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": не удалось удалить правую ценовую метку! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Проверяет значения точки привязки и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- если время точки не задано, то она будет на текущем баре
if(!time)
    time=TimeCurrent();
}

```

```

//--- если цена точки не задана, то она будет иметь значение Bid
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//+-----+
//| Script program start function | 
//+-----+

void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}

//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

//--- размер массива price
int accuracy=1000;

//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки метки
datetime date[];
double price[];

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);

//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);

//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;

//--- определим точки для рисования метки
int d=InpDate*(bars-1)/100;
int p=InpPrice*(accuracy-1)/100;

//--- создадим правую ценовую метку на графике
if(!ArrowRightPriceCreate(0,InpName,0,date[d],price[p],InpColor,
    InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}

```

```
}

//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- теперь будем перемещать точку привязки
//--- счетчик цикла
int v_steps=accuracy*4/5;
//--- перемещаем точку привязки
for(int i=0;i<v_steps;i++)
{
    //--- возьмем следующее значение
    if(p>1)
        p-=1;
    //--- сдвигаем точку
    if(!ArrowRightPriceMove(0,IInpName,date[d],price[p]))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
}

//--- задержка в 1 секунду
Sleep(1000);

//--- удалим метку с графика
ArrowRightPriceDelete(0,IInpName);
ChartRedraw();

//--- задержка в 1 секунду
Sleep(1000);

//---
```

## OBJ\_ARROW\_BUY

Знак "Buy".



### Пример

Следующий скрипт создает и перемещает на графике значки "Buy". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует значки \"Buy\" в окне графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input color InpColor=C'3,95,172'; // Цвет значков
//+
//| Создает знак "Buy"
//+
bool ArrowBuyCreate(const long
                     const string
                     const int
                     datetime
                     double
                     const color
                     const ENUM_LINE_STYLE style=STYLE_SOLID,
                     const int
                     const bool
                     const bool
                     chart_ID=0,           // ID графика
                     name="ArrowBuy",      // имя знака
                     sub_window=0,         // номер подокна
                     time=0,              // время точки привязки
                     price=0,             // цена точки привязки
                     clr=C'3,95,172',      // цвет знака
                     width=1,              // стиль линии (при выде
                     back=false,           // размер линии (при выде
                     selection=false);    // на заднем плане
                     // выделить для перемещен
```

```

        const bool           hidden=true,          // скрыт в списке объектов
        const long            z_order=0)           // приоритет на нажатие мыши

    {

//--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);

//--- сбросим значение ошибки
    ResetLastError();

//--- создадим знак
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_BUY,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать знак \"Buy\"! Код ошибки = ",GetLastError());
        return(false);
    }

//--- установим цвет знака
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);

//--- установим стиль линии (при выделении)
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);

//--- установим размер линии (при выделении)
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);

//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);

//--- включим (true) или отключим (false) режим перемещения знака мышью
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

//--- установи приоритет на получение события нажатия мыши на графике
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

//--- успешное выполнение
    return(true);
}

//----------------------------------------------------------------------------------------------------------------+
//| Перемещает точку привязки                                |
//----------------------------------------------------------------------------------------------------------------+
bool ArrowBuyMove(const long   chart_ID=0,           // ID графика
                  const string name="ArrowBuy", // имя объекта
                  datetime    time=0,          // координата времени точки привязки
                  double      price=0)         // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);

//--- сбросим значение ошибки
    ResetLastError();

//--- переместим точку привязки
    if(!ObjectMove(chart_ID,name,0,time,price))

```

```

{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ", GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет знак "Buy" |
//+-----+
bool ArrowBuyDelete(const long chart_ID=0,           // ID графика
                     const string name="ArrowBuy") // имя знака
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- удалим знак
    if(!ObjectDelete(chart_ID, name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить знак \"Buy\"! Код ошибки = ", GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

//+-----+
//| Проверяет значения точки привязки и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
    //--- если время точки не задано, то она будет на текущем баре
    if(!time)
        time=TimeCurrent();
    //--- если цена точки не задана, то она будет иметь значение Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime date[]; // массив для хранения дат видимых баров
    double low[]; // массив для хранения цен Low видимых баров
    double high[]; // массив для хранения цен High видимых баров
    //--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
}

```

```

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(low,bars);
ArrayResize(high,bars);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен Low
if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
{
    Print("Не удалось скопировать значения цен Low! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен High
if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
{
    Print("Не удалось скопировать значения цен High! Код ошибки = ",GetLastError());
    return;
}
//--- создадим значки "Buy" в точке Low для каждого видимого бара
for(int i=0;i<bars;i++)
{
    if(!ArrowBuyCreate(0,"ArrowBuy_"+(string)i,0,date[i],low[i],InpColor))
        return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}
//--- переместим значки "Buy" в точку High для каждого видимого бара
for(int i=0;i<bars;i++)
{
    if(!ArrowBuyMove(0,"ArrowBuy_"+(string)i,date[i],high[i]))
        return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}

```

```
//--- удалим значки "Buy"
for(int i=0;i<bars;i++)
{
    if(!ArrowBuyDelete(0,"ArrowBuy_"+(string)i))
        return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}
//---
```

## OBJ\_ARROW\_SELL

Знак "Sell".



### Пример

Следующий скрипт создает и перемещает на графике значки "Sell". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует значки \"Sell\" в окне графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input color InpColor=C'225,68,29'; // Цвет значков
//+
//| Создает знак "Sell"
//+
bool ArrowSellCreate(const long           chart_ID=0,          // ID графика
                     const string        name="ArrowSell", // имя знака
                     const int           sub_window=0,    // номер подокна
                     const datetime      time=0,         // время точки привязки
                     double              price=0,        // цена точки привязки
                     const color          clr=C'225,68,29', // цвет знака
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии (при выде
                     const int           width=1,        // размер линии (при выд
                     const bool          back=false,     // на заднем плане
                     const bool          selection=false, // выделить для перемеще
```

```

        const bool           hidden=true,          // скрыт в списке объектов
        const long            z_order=0)           // приоритет на нажатие

    {

//--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);

//--- сбросим значение ошибки
    ResetLastError();

//--- создадим знак
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW_SELL,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать знак \"Sell\"! Код ошибки = ",GetLastError());
        return(false);
    }

//--- установим цвет знака
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);

//--- установим стиль линии (при выделении)
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);

//--- установим размер линии (при выделении)
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);

//--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);

//--- включим (true) или отключим (false) режим перемещения знака мышью
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
    ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

//--- установим приоритет на получение события нажатия мыши на графике
    ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

//--- успешное выполнение
    return(true);
}

//+-----+
//| Перемещает точку привязки |
//+-----+
bool ArrowSellMove(const long   chart_ID=0,          // ID графика
                   const string name="ArrowSell", // имя объекта
                   datetime    time=0,           // координата времени точки привязки
                   double      price=0)         // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
    if(!time)
        time=TimeCurrent();
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);

//--- сбросим значение ошибки
    ResetLastError();

//--- переместим точку привязки
    if(!ObjectMove(chart_ID,name,0,time,price))

```

```

{
    Print(__FUNCTION__,
        ": не удалось переместить точку привязки! Код ошибки = ", GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет знак "Sell" |
//+-----+
bool ArrowSellDelete(const long chart_ID=0,           // ID графика
                     const string name="ArrowSell") // имя знака
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- удалим знак
    if(!ObjectDelete(chart_ID, name))
    {
        Print(__FUNCTION__,
            ": не удалось удалить знак \"Sell\"! Код ошибки = ", GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

//+-----+
//| Проверяет значения точки привязки, и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+
void ChangeArrowEmptyPoint(datetime &time,double &price)
{
    //--- если время точки не задано, то она будет на текущем баре
    if(!time)
        time=TimeCurrent();
    //--- если цена точки не задана, то она будет иметь значение Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    datetime date[]; // массив для хранения дат видимых баров
    double low[]; // массив для хранения цен Low видимых баров
    double high[]; // массив для хранения цен High видимых баров
    //--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
}

```

```

//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(low,bars);
ArrayResize(high,bars);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен Low
if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
{
    Print("Не удалось скопировать значения цен Low! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен High
if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
{
    Print("Не удалось скопировать значения цен High! Код ошибки = ",GetLastError());
    return;
}
//--- создадим значки "Sell" в точке High для каждого видимого бара
for(int i=0;i<bars;i++)
{
    if(!ArrowSellCreate(0,"ArrowSell_"+(string)i,0,date[i],high[i],InpColor))
        return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}
//--- переместим значки "Sell" в точку Low для каждого видимого бара
for(int i=0;i<bars;i++)
{
    if(!ArrowSellMove(0,"ArrowSell_"+(string)i,date[i],low[i]))
        return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}

```

```
//--- удалим значки "Sell"
for(int i=0;i<bars;i++)
{
    if(!ArrowSellDelete(0,"ArrowSell_"+(string)i))
        return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}
//---
```

## OBJ\_ARROW

Объект "Стрелка".



### Примечание

Положение точки привязки относительно стрелки можно выбрать из перечисления [ENUM\\_ARROW\\_ANCHOR](#).

Стрелки большого размера (больше 5) можно создать только установив соответствующее значение свойства [OBJPROP\\_WIDTH](#) при написании кода в MetaEditor.

Нужный тип стрелки можно выбрать, установив один из кодов символов шрифта [Wingdings](#).

### Пример

Следующий скрипт создает на графике объект "Стрелка" и изменяет ее тип. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт создает произвольную стрелку в окне графика."
#property description "Координата точки привязки задается в процентах"
#property description "от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="Arrow";           // Имя стрелки
input int              InpDate=50;                // Дата точки привязки в %
input int              InpPrice=50;               // Цена точки привязки в %

```

```

input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Способ привязки
input color InpColor=clrDodgerBlue; // Цвет стрелки
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID; // Стиль окаймляющей линии
input int InpWidth=10; // Размер стрелки
input bool InpBack=false; // Стрелка на заднем плане
input bool InpSelection=false; // Выделить для перемещений
input bool InpHidden=true; // Скрыт в списке объектов
input long InpZOrder=0; // Приоритет на нажатие мышью
//+-----+
//| Создает стрелку |
//+-----+
bool ArrowCreate(const long chart_ID=0, // ID графика
                 const string name="Arrow", // имя стрелки
                 const int sub_window=0, // номер подокна
                 datetime time=0, // время точки привязки
                 double price=0, // цена точки привязки
                 const uchar arrow_code=252, // код стрелки
                 const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // положение точки привязки
                 const color clr=clrRed, // цвет стрелки
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль окаймляющей линии
                 const int width=3, // размер стрелки
                 const bool back=false, // на заднем плане
                 const bool selection=true, // выделить для перемещения
                 const bool hidden=true, // скрыт в списке объектов
                 const long z_order=0) // приоритет на нажатие
{
    //--- установим координаты точки привязки, если они не заданы
    ChangeArrowEmptyPoint(time,price);
    //---бросим значение ошибки
    ResetLastError();
    //--- создадим стрелку
    if(!ObjectCreate(chart_ID,name,OBJ_ARROW,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать стрелку! Код ошибки = ",GetLastError());
        return(false);
    }
    //--- установим код стрелки
    ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,arrow_code);
    //--- установим способ привязки
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
    //--- установим цвет стрелки
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- установим стиль окаймляющей линии
    ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
    //--- установим размер стрелки
    ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
    //--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
}

```

```

//--- включим (true) или отключим (false) режим перемещения стрелки мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки |
//+-----+
bool ArrowMove(const long    chart_ID=0,      // ID графика
               const string name="Arrow", // имя объекта
               datetime     time=0,       // координата времени точки привязки
               double       price=0)     // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);

//--- сбросим значение ошибки
ResetLastError();

//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Меняет код стрелки |
//+-----+
bool ArrowCodeChange(const long    chart_ID=0,      // ID графика
                     const string name="Arrow", // имя объекта
                     const uchar   code=252)    // код стрелки
{
//--- сбросим значение ошибки
ResetLastError();

//--- изменим код стрелки
if(!ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,code))

```

```

{
    Print(__FUNCTION__,
          ": не удалось изменить код стрелки! Код ошибки = ", GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
// | Меняет способ привязки
//+-----+
bool ArrowAnchorChange(const long           chart_ID=0,           // ID графика
                       const string        name="Arrow",        // имя объекта
                       const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ привязки
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- изменим способ привязки
    if(!ObjectSetInteger(chart_ID, name, OBJPROP_ANCHOR, anchor))
    {
        Print(__FUNCTION__,
              ": не удалось изменить способ привязки! Код ошибки = ", GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

//+-----+
// | Удаляет стрелку
//+-----+
bool ArrowDelete(const long   chart_ID=0,   // ID графика
                 const string name="Arrow") // имя стрелки
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- удалим стрелку
    if(!ObjectDelete(chart_ID, name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить стрелку! Код ошибки = ", GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

//+-----+
// | Проверяет значения точки привязки и для пустых значений
// | устанавливает значения по умолчанию
//+-----+

```

```

void ChangeArrowEmptyPoint(datetime &time,double &price)
{
//--- если время точки не задано, то она будет на текущем баре
if(!time)
    time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}

//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
datetime date[];
double price[];
//--- выделение памяти
ArrayResize(date,bars);
ArrayResize(price,accuracy);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}

//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
double step=(max_price-min_price)/accuracy;
for(int i=0;i<accuracy;i++)
    price[i]=min_price+i*step;
//--- определим точки для рисования стрелки
int d=InpDate*(bars-1)/100;
int p=InpPrice*(accuracy-1)/100;
}

```

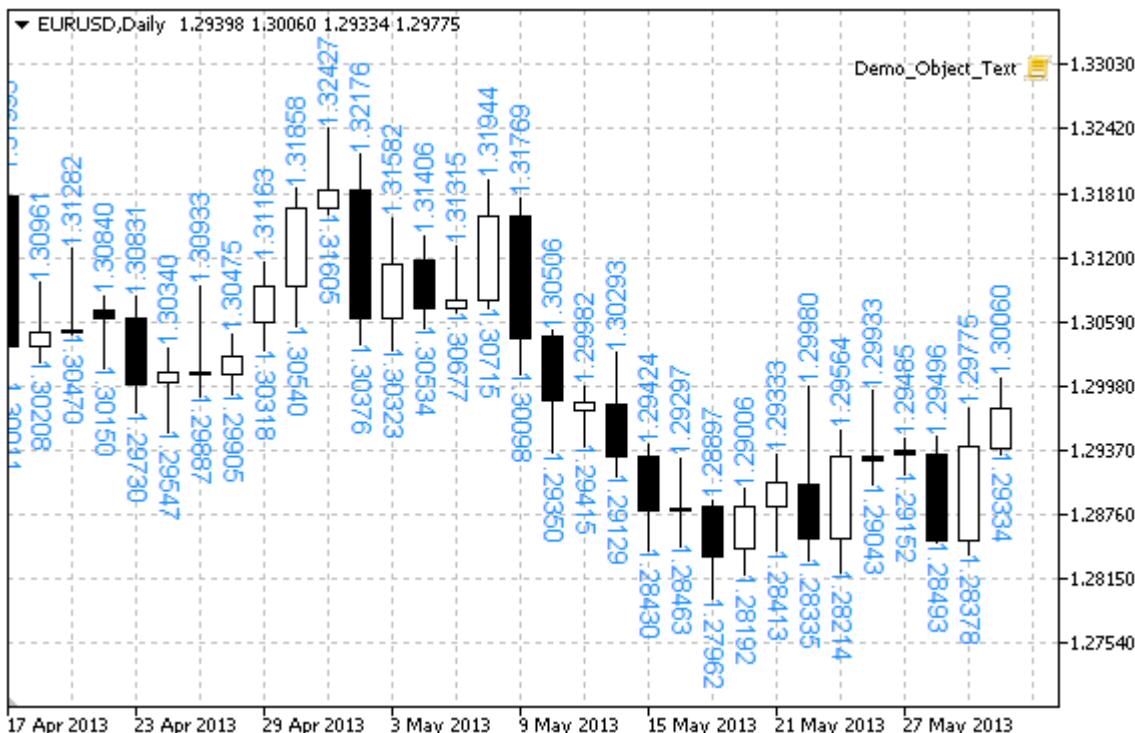
```
//--- создадим стрелку на графике
if(!ArrowCreate(0,InpName,0,date[d],price[p],32,InpAnchor,InpColor,
InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}

//--- перерисуем график
ChartRedraw();

//--- в цикле рассмотрим все варианты построения стрелок
for(int i=33;i<256;i++)
{
    if(!ArrowCodeChange(0,InpName,(uchar)i))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в полсекунды
    Sleep(500);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим стрелку с графика
ArrowDelete(0,InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_TEXT

Объект "Текст".



### Примечание

Положение точки привязки относительно текста можно выбрать из перечисления [ENUM\\_ANCHOR\\_POINT](#). Также можно менять угол наклона текста при помощи свойства [OBJPROP\\_ANGLE](#).

### Пример

Следующий скрипт создает на графике несколько объектов "Текст". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает графический объект \"Текст\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpFont="Arial";           // Шрифт
input int                InpFontSize=10;          // Размер шрифта
input color              InpColor=clrRed;         // Цвет
input double             InpAngle=90.0;          // Угол наклона в градусах
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_LEFT;   // Способ привязки
input bool               InpBack=false;          // Объект на заднем плане
input bool               InpSelection=false;       // Выделить для перемещений
input bool               InpHidden=true;          // Скрыт в списке объектов
```

```

input long           InpZOrder=0;           // Приоритет на нажатие мышью
//+-----+
//| Создает объект "Текст"
//+-----+
bool TextCreate(const long          chart_ID=0,           // ID графика
                const string        name="Text",           // имя объекта
                const int            sub_window=0,         // номер подокна
                datetime            time=0,              // время точки привязки
                double               price=0,             // цена точки привязки
                const string         text="Text",           // сам текст
                const string         font="Arial",          // шрифт
                const int            font_size=10,          // размер шрифта
                const color           clr=clrRed,          // цвет
                const double          angle=0.0,            // наклон текста
                const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // способ привязки
                const bool             back=false,           // на заднем плане
                const bool             selection=false,       // выделить для перевода в другой режим
                const bool             hidden=true,           // скрыт в списке объектов
                const long             z_order=0)           // приоритет на нажатие мышью
{
    //--- установим координаты точки привязки, если они не заданы
    ChangeTextEmptyPoint(time,price);
    //--- сбросим значение ошибки
    ResetLastError();
    //--- создадим объект "Текст"
    if(!ObjectCreate(chart_ID,name,OBJ_TEXT,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать объект \"Текст\"! Код ошибки = ",GetLastError());
        return(false);
    }
    //--- установим текст
    ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
    //--- установим шрифт текста
    ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
    //--- установим размер шрифта
    ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
    //--- установим угол наклона текста
    ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
    //--- установим способ привязки
    ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
    //--- установим цвет
    ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
    //--- отобразим на переднем (false) или заднем (true) плане
    ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
    //--- включим (true) или отключим (false) режим перемещения объекта мышью
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
    ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
    //--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
}

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает точку привязки |
//+-----+
bool TextMove(const long    chart_ID=0,   // ID графика
              const string name="Text", // имя объекта
              datetime      time=0,     // координата времени точки привязки
              double        price=0)   // координата цены точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//---бросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
          ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет текст объекта |
//+-----+
bool TextChange(const long    chart_ID=0,   // ID графика
                const string name="Text", // имя объекта
                const string text="Text") // текст
{
//---бросим значение ошибки
ResetLastError();
//---изменим текст объекта
if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
{
    Print(__FUNCTION__,
          ": не удалось изменить текст! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

```

```

    }

//+-----+
//| Удаляет объект "Текст" |
//+-----+

bool TextDelete(const long    chart_ID=0, // ID графика
                const string name="Text") // имя объекта
{
    //--- сбросим значение ошибки
    ResetLastError();

    //--- удалим объект
    if(!ObjectDelete(chart_ID, name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить объект \"Текст\"! Код ошибки = ", GetLastError());
        return(false);
    }

    //--- успешное выполнение
    return(true);
}

//+-----+
//| Проверяет значения точки привязки и для пустых значений |
//| устанавливает значения по умолчанию |
//+-----+

void ChangeTextEmptyPoint(datetime &time,double &price)
{
    //--- если время точки не задано, то она будет на текущем баре
    if(!time)
        time=TimeCurrent();

    //--- если цена точки не задана, то она будет иметь значение Bid
    if(!price)
        price=SymbolInfoDouble(Symbol(),SYMBOL_BID);

}

//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
    datetime date[]; // массив для хранения дат видимых баров
    double    low[]; // массив для хранения цен Low видимых баров
    double    high[]; // массив для хранения цен High видимых баров

    //--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);

    //--- выделение памяти
    ArrayResize(date,bars);
    ArrayResize(low,bars);
    ArrayResize(high,bars);

    //--- заполним массив дат
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
}

```

```

{
    Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен Low
if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
{
    Print("Не удалось скопировать значения цен Low! Код ошибки = ",GetLastError());
    return;
}
//--- заполним массив цен High
if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
{
    Print("Не удалось скопировать значения цен High! Код ошибки = ",GetLastError());
    return;
}
//--- определим, как часто надо делать надписи
int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- определим шаг
int step=1;
switch(scale)
{
    case 0:
        step=12;
        break;
    case 1:
        step=6;
        break;
    case 2:
        step=4;
        break;
    case 3:
        step=2;
        break;
}
//--- создадим надписи для значений High и Low баров (с промежутками)
for(int i=0;i<bars;i+=step)
{
    //--- создаем надписи
    if(!TextCreate(0,"TextHigh_"+(string)i,0,date[i],high[i],DoubleToString(high[i],
        InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder)))
    {
        return;
    }
    if(!TextCreate(0,"TextLow_"+(string)i,0,date[i],low[i],DoubleToString(low[i],5),
        InpColor,-InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
    {
        return;
    }
}

```

```
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}

//--- задержка в полсекунды
Sleep(500);
//--- удалим надписи
for(int i=0;i<bars;i+=step)
{
    if(!TextDelete(0,"TextHigh_"+(string)i))
        return;
    if(!TextDelete(0,"TextLow_"+(string)i))
        return;
    //--- перерисуем график
    ChartRedraw();
    // задержка в 0.05 секунды
    Sleep(50);
}
}

//---
```

## OBJ\_LABEL

Объект "Текстовая метка".



### Примечание

Положение точки привязки относительно метки можно выбрать из перечисления [ENUM\\_ANCHOR\\_POINT](#). Координаты точки привязки задаются в пикселях.

Также можно выбрать угол привязки текстовой метки из перечисления [ENUM\\_BASE\\_CORNER](#).

### Пример

Следующий скрипт создает и перемещает на графике объект "Текстовая метка". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает графический объект \"Текстовая метка\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="Label";          // Имя метки
input int              InpX=150;                 // Расстояние по оси X
input int              InpY=150;                 // Расстояние по оси Y
input string           InpFont="Arial";           // Шрифт
input int              InpFontSize=14;            // Размер шрифта
input color            InpColor=clrRed;           // Цвет
input double           InpAngle=0.0;               // Угол наклона в градусах
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_CENTER; // Способ привязки
```

```

input bool           InpBack=false;           // Объект на заднем плане
input bool           InpSelection=true;        // Выделить для перемещений
input bool           InpHidden=true;          // Скрыт в списке объектов
input long           InpZOrder=0;            // Приоритет на нажатие мышью
//+-----+
//| Создает текстовую метку
//+-----+
bool LabelCreate(const long           chart_ID=0,           // ID графика
                  const string        name="Label",         // имя метки
                  const int            sub_window=0,        // номер подокна
                  const int            x=0,                 // координата по оси X
                  const int            y=0,                 // координата по оси Y
                  const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // угол графика для
                  const string        text="Label",         // текст
                  const string        font="Arial",         // шрифт
                  const int            font_size=10,        // размер шрифта
                  const color          clr=clrRed,          // цвет
                  const double         angle=0.0,          // наклон текста
                  const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // способ привязки
                  const bool            back=false,          // на заднем плане
                  const bool            selection=false,      // выделить для перемещения
                  const bool            hidden=true,          // скрыт в списке объектов
                  const long            z_order=0)           // приоритет на нажатие
{
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим текстовую метку
    if(!ObjectCreate(chart_ID, name, OBJ_LABEL, sub_window, 0, 0))
    {
        Print(__FUNCTION__,
              ": не удалось создать текстовую метку! Код ошибки = ", GetLastError());
        return(false);
    }
//--- установим координаты метки
    ObjectSetInteger(chart_ID, name, OBJPROP_XDISTANCE, x);
    ObjectSetInteger(chart_ID, name, OBJPROP_YDISTANCE, y);
//--- установим угол графика, относительно которого будут определяться координаты точек
    ObjectSetInteger(chart_ID, name, OBJPROP_CORNER, corner);
//--- установим текст
    ObjectSetString(chart_ID, name, OBJPROP_TEXT, text);
//--- установим шрифт текста
    ObjectSetString(chart_ID, name, OBJPROP_FONT, font);
//--- установим размер шрифта
    ObjectSetInteger(chart_ID, name, OBJPROP_FONTSIZE, font_size);
//--- установим угол наклона текста
    ObjectSetDouble(chart_ID, name, OBJPROP_ANGLE, angle);
//--- установим способ привязки
    ObjectSetInteger(chart_ID, name, OBJPROP_ANCHOR, anchor);
//--- установим цвет

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает текстовую метку |
//+-----+
bool LabelMove(const long chart_ID=0, // ID графика
               const string name="Label", // имя метки
               const int x=0,           // координата по оси X
               const int y=0)           // координата по оси Y
{
//--- сбросим значение ошибки
ResetLastError();
//--- переместим текстовую метку
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
{
Print(__FUNCTION__,
      ": не удалось переместить X-координату метки! Код ошибки = ",GetLastError
      return(false);
}
if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
{
Print(__FUNCTION__,
      ": не удалось переместить Y-координату метки! Код ошибки = ",GetLastError
      return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет угол графика для привязки метки |
//+-----+
bool LabelChangeCorner(const long chart_ID=0, // ID графика
                      const string name="Label", // имя метки
                      const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) // угол графика
{
//--- сбросим значение ошибки
ResetLastError();
//--- изменим угол привязки

```

```

if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
{
    Print(__FUNCTION__,
          ": не удалось изменить угол привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет текст метки |
//+-----+
bool LabelTextChange(const long    chart_ID=0,      // ID графика
                     const string name="Label", // имя объекта
                     const string text="Text") // текст
{
//--- сбросим значение ошибки
ResetLastError();
//--- изменим текст объекта
if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
{
    Print(__FUNCTION__,
          ": не удалось изменить текст! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет текстовую метку |
//+-----+
bool LabelDelete(const long    chart_ID=0,      // ID графика
                 const string name="Label") // имя метки
{
//--- сбросим значение ошибки
ResetLastError();
//--- удалим метку
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": не удалось удалить текстовую метку! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Script program start function |
//+-----+

```

```

void OnStart()
{
//--- запомним координаты метки в локальные переменные
int x=InpX;
int y=InpY;
//--- размеры окна графика
long x_distance;
long y_distance;
//--- определим размеры окна
if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
{
Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
return;
}
if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
{
Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
return;
}
//--- проверим входные параметры на корректность
if(InpX<0 || InpX>x_distance-1 || InpY<0 || InpY>y_distance-1)
{
Print("Ошибка! Некорректные значения входных параметров!");
return;
}
//--- подготовим начальный текст для метки
string text;
StringConcatenate(text,"Левый верхний угол: ",x,",",y);
//--- создадим текстовую метку на графике
if(!LabelCreate(0,InpName,0,InpX,InpY,CORNER_LEFT_UPPER,text,InpFont,InpFontSize,
InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
{
return;
}
//--- перерисуем график и подождем полсекунды
ChartRedraw();
Sleep(500);
//--- будем перемещать метку и одновременно менять ее текст
//--- количество итераций по осям
int h_steps=(int)(x_distance/2-InpX);
int v_steps=(int)(y_distance/2-InpY);
//--- переместим метку вниз
for(int i=0;i<v_steps;i++)
{
//--- меняем координату
y+=2;
//--- перемещаем метку и меняем ее текст
MoveAndTextChange(x,y,"Левый верхний угол: ");
}
}

```

```

//--- задержка в полсекунды
Sleep(500);
//--- переместим метку вправо
for(int i=0;i<h_steps;i++)
{
    //--- меняем координату
    x+=2;
    //--- перемещаем метку и меняем ее текст
    MoveAndTextChange(x,y,"Левый верхний угол: ");
}
//--- задержка в полсекунды
Sleep(500);
//--- переместим метку вверх
for(int i=0;i<v_steps;i++)
{
    //--- меняем координату
    y-=2;
    //--- перемещаем метку и меняем ее текст
    MoveAndTextChange(x,y,"Левый верхний угол: ");
}
//--- задержка в полсекунды
Sleep(500);
//--- переместим метку влево
for(int i=0;i<h_steps;i++)
{
    //--- меняем координату
    x-=2;
    //--- перемещаем метку и меняем ее текст
    MoveAndTextChange(x,y,"Левый верхний угол: ");
}
//--- задержка в полсекунды
Sleep(500);
//--- теперь переместим точку путем изменения угла привязки
//--- переместим в левый нижний угол
if(!LabelChangeCorner(0,InpName,CORNER_LEFT_LOWER))
    return;
//--- изменим текст метки
StringConcatenate(text,"Левый нижний угол: ",x,",",y);
if(!LabelTextChange(0,InpName,text))
    return;
//--- перерисуем график и подождем две секунды
ChartRedraw();
Sleep(2000);
//--- переместим в правый нижний угол
if(!LabelChangeCorner(0,InpName,CORNER_RIGHT_LOWER))
    return;
//--- изменим текст метки
StringConcatenate(text,"Правый нижний угол: ",x,",",y);
if(!LabelTextChange(0,InpName,text))

```

```

        return;
//--- перерисуем график и подождем две секунды
    ChartRedraw();
    Sleep(2000);
//--- переместим в правый верхний угол
    if(!LabelChangeCorner(0, InpName, CORNER_RIGHT_UPPER))
        return;
//--- изменим текст метки
    StringConcatenate(text,"Правый верхний угол: ",x,",",y);
    if(!LabelTextChange(0, InpName, text))
        return;
//--- перерисуем график и подождем две секунды
    ChartRedraw();
    Sleep(2000);
//--- переместим в левый верхний угол
    if(!LabelChangeCorner(0, InpName, CORNER_LEFT_UPPER))
        return;
//--- изменим текст метки
    StringConcatenate(text,"Левый верхний угол: ",x,",",y);
    if(!LabelTextChange(0, InpName, text))
        return;
//--- перерисуем график и подождем две секунды
    ChartRedraw();
    Sleep(2000);
//--- удалим метку
    LabelDelete(0, InpName);
//--- перерисуем график и подождем полсекунды
    ChartRedraw();
    Sleep(500);
//---
}

//-----+
//| Функция перемещает объект и меняет его текст |
//-----+
bool MoveAndTextChange(const int x,const int y,string text)
{
//--- перемещаем метку
    if(!LabelMove(0, InpName,x,y))
        return(false);
//--- изменим текст метки
    StringConcatenate(text,text,x,",",y);
    if(!LabelTextChange(0, InpName, text))
        return(false);
//--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return(false);
//--- перерисуем график
    ChartRedraw();
// задержка в 0.01 секунды
}

```

```
    Sleep(10);
//--- выход из функции
    return(true);
}
```

## OBJ\_BUTTON

Объект "Кнопка".



### Примечание

Координаты точки привязки задаются в пикселях. Можно выбрать угол привязки кнопки из перечисления [ENUM\\_BASE\\_CORNER](#).

### Пример

Следующий скрипт создает и перемещает на графике объект "Кнопка". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
---- описание
#property description "Скрипт создает кнопку на графике."
---- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
---- входные параметры скрипта
input string           InpName="Button";                      // Имя кнопки
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER;          // Угол графика для привязки
input string           InpFont="Arial";                        // Шрифт
input int              InpFontSize=14;                         // Размер шрифта
input color            InpColor=clrBlack;                      // Цвет текста
input color            InpBackColor=C'236,233,216';          // Цвет фона
input color            InpBorderColor=clrNONE;                  // Цвет границы
input bool             InpState=false;                        // Нажата/Отжата
input bool             InpBack=false;                         // Объект на заднем плане
input bool             InpSelection=false;                    // Выделить для перемещений
```

```

input bool           InpHidden=true;           // Скрыт в списке объектов
input long          InpZOrder=0;             // Приоритет на нажатие мышью
//+
//| Создает кнопку
//+
bool ButtonCreate(const long           chart_ID=0,           // ID графика
                   const string        name="Button",        // имя кнопки
                   const int            sub_window=0,       // номер подокна
                   const int            x=0,                 // координата по оси X
                   const int            y=0,                 // координата по оси Y
                   const int            width=50,            // ширина кнопки
                   const int            height=18,           // высота кнопки
                   const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // угол графика для
                                                               // создания
                   const string        text="Button",        // текст
                   const string        font="Arial",         // шрифт
                   const int            font_size=10,          // размер шрифта
                   const color          clr=clrBlack,         // цвет текста
                   const color          back_clr=C'236,233,216', // цвет фона
                   const color          border_clr=clrNONE,    // цвет границы
                   const bool           state=false,          // нажата/отжата
                   const bool           back=false,           // на заднем плане
                   const bool           selection=false,      // выделить для переноса
                   const bool           hidden=true,          // скрыт в списке
                   const long           z_order=0)           // приоритет на нажатие
{
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим кнопку
    if(!ObjectCreate(chart_ID,name,OBJ_BUTTON,sub_window,0,0))
    {
        Print(__FUNCTION__,
              ": не удалось создать кнопку! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим координаты кнопки
    ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
    ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- установим размер кнопки
    ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
    ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим угол графика, относительно которого будут определяться координаты точек
    ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- установим текст
    ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- установим шрифт текста
    ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- установим размер шрифта
    ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- установим цвет текста

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим цвет фона
ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- установим цвет границы
ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- переведем кнопку в заданное состояние
ObjectSetInteger(chart_ID,name,OBJPROP_STATE,state);
//--- включим (true) или отключим (false) режим перемещения кнопки мышью
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает кнопку |
//+-----+
bool ButtonMove(const long chart_ID=0, // ID графика
                const string name="Button", // имя кнопки
                const int x=0, // координата по оси X
                const int y=0) // координата по оси Y
{
//--- сбросим значение ошибки
ResetLastError();
//--- переместим кнопку
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
{
Print(__FUNCTION__,
      ": не удалось переместить X-координату кнопки! Код ошибки = ",GetLastError());
return(false);
}
if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
{
Print(__FUNCTION__,
      ": не удалось переместить Y-координату кнопки! Код ошибки = ",GetLastError());
return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет размер кнопки |
//+-----+
bool ButtonChangeSize(const long chart_ID=0, // ID графика
                      const int width=0, // ширина
                      const int height=0) // высота
{
//--- установим ширину и высоту
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
ObjectSetInteger(chart_ID,name,OBJPROP_HEIGHT,height);
//--- успешное выполнение
return(true);
}

```

```

        const string name="Button", // имя кнопки
        const int     width=50,      // ширина кнопки
        const int     height=18)    // высота кнопки

    {
//--- сбросим значение ошибки
    ResetLastError();
//--- изменим размеры кнопки
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
    {
        Print(__FUNCTION__,
            ": не удалось изменить ширину кнопки! Код ошибки = ",GetLastError());
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
    {
        Print(__FUNCTION__,
            ": не удалось изменить высоту кнопки! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

//+-----+
//| Изменяет угол графика для привязки кнопки |
//+-----+
bool ButtonChangeCorner(const long           chart_ID=0,          // ID графика
                       const string       name="Button",        // имя кнопки
                       const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) // угол графика
{
//--- сбросим значение ошибки
    ResetLastError();
//--- изменим угол привязки
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
    {
        Print(__FUNCTION__,
            ": не удалось изменить угол привязки! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}

//+-----+
//| Изменяет текст кнопки |
//+-----+
bool ButtonTextChange(const long   chart_ID=0,      // ID графика
                      const string name="Button", // имя кнопки
                      const string text="Text") // текст
{
//--- сбросим значение ошибки

```

```

        ResetLastError();

//--- изменим текст объекта
if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
{
    Print(__FUNCTION__,
          ": не удалось изменить текст! Код ошибки = ",GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Удаляет кнопку |
//+-----+

bool ButtonDelete(const long    chart_ID=0,      // ID графика
                  const string name="Button") // имя кнопки
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- удалим кнопку
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить кнопку! Код ошибки = ",GetLastError());
        return(false);
    }

    //--- успешное выполнение
    return(true);
}

//+-----+
//| Script program start function |
//+-----+

void OnStart()
{
    //--- размеры окна графика
    long x_distance;
    long y_distance;
    //--- определим размеры окна
    if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
    {
        Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
        return;
    }
    if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
    {
        Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
        return;
    }

    //--- определим шаг для изменения размеров кнопки
}

```

```

int x_step=(int)x_distance/32;
int y_step=(int)y_distance/32;
//--- установим координаты кнопки и ее размер
int x=(int)x_distance/32;
int y=(int)y_distance/32;
int x_size=(int)x_distance*15/16;
int y_size=(int)y_distance*15/16;
//--- создадим кнопку
if(!ButtonCreate(0,InpName,0,x,y,x_size,y_size,InpCorner,"Press",InpFont,InpFontSize,
InpColor,InpBackColor,InpBorderColor,InpState,InpBack,InpSelection,InpHidden,Inp
{
    return;
}
//--- перерисуем график
ChartRedraw();
//--- в цикле уменьшаем кнопку
int i=0;
while(i<13)
{
    //--- задержка в полсекунды
    Sleep(500);
    //--- переведем кнопку в нажатое состояние
    ObjectSetInteger(0,InpName,OBJPROP_STATE,true);
    //--- перерисуем график и подождем 0.2 секунды
    ChartRedraw();
    Sleep(200);
    //--- переопределим координаты и размер кнопки
    x+=x_step;
    y+=y_step;
    x_size-=x_step*2;
    y_size-=y_step*2;
    //--- уменьшим кнопку
    ButtonMove(0,InpName,x,y);
    ButtonChangeSize(0,InpName,x_size,y_size);
    //--- вернем кнопку в ненажатое состояние
    ObjectSetInteger(0,InpName,OBJPROP_STATE,false);
    //--- перерисуем график
    ChartRedraw();
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- увеличим счетчик цикла
    i++;
}
//--- задержка в полсекунды
Sleep(500);
//--- удалим кнопку
ButtonDelete(0,InpName);
ChartRedraw();

```

```
//--- подождем 1 секунду  
Sleep(1000);  
//---  
}
```

## OBJ\_CHART

Объект "График".



### Примечание

Координаты точки привязки задаются в пикселях. Можно выбрать угол привязки из перечисления [ENUM\\_BASE\\_CORNER](#).

Для объекта "График" можно выбирать символ, период и масштаб, а также включать/отключать режим отображения шкалы цены и даты.

### Пример

Следующий скрипт создает и перемещает на графике объект "График". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт создает объект \"График\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="Chart";           // Имя объекта
input string           InpSymbol="EURUSD";         // Символ
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H1;       // Период
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Угол для привязки
input int              InpScale=2;                // Масштаб
input bool             InpDateScale=true;          // Отображение шкалы времени
input bool             InpPriceScale=true;          // Отображение шкалы цены

```

```

input color           InpColor=clrRed;           // Цвет рамки при выделении
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линии при выделении
input int             InpPointWidth=1;          // Размер точки для перемещений
input bool            InpBack=false;            // Объект на заднем плане
input bool            InpSelection=true;          // Выделить для перемещений
input bool            InpHidden=true;            // Скрыт в списке объектов
input long            InpZOrder=0;               // Приоритет на нажатие мышью
//+-----+
//| Создает объект "График" |
//+-----+
bool ObjectChartCreate(const long           chart_ID=0,           // ID графика
                       const string        name="Chart",         // имя объекта
                       const int            sub_window=0,        // номер подокна
                       const string        symbol="EURUSD",       // символ
                       const ENUM_TIMEFRAMES period=PERIOD_H1, // период
                       const int            x=0,                // координата X
                       const int            y=0,                // координата Y
                       const int            width=300,          // ширина
                       const int            height=200,         // высота
                       const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // угол для границы
                       const int            scale=2,             // масштаб
                       const bool           date_scale=true,      // отображение даты
                       const bool           price_scale=true,     // отображение цен
                       const color          clr=clrRed,          // цвет рамки
                       const ENUM_LINE_STYLE style=STYLE_SOLID,    // стиль линии
                       const int            point_width=1,        // размер точки
                       const bool           back=false,          // на заднем плане
                       const bool           selection=false,     // выделить для перемещения
                       const bool           hidden=true,          // скрыт в списке
                       const long            z_order=0)           // приоритет

{
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим объект "График"
    if(!ObjectCreate(chart_ID,name,OBJ_CHART,sub_window,0,0))
    {
        Print(__FUNCTION__,
              ": не удалось создать объект \"График\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим координаты объекта
    ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
    ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- установим размер объекта
    ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
    ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим угол графика, относительно которого будут определяться координаты точек
    ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- установим символ
}

```

```

ObjectSetString(chart_ID,name,OBJPROP_SYMBOL,symbol);
//--- установим период
ObjectSetInteger(chart_ID,name,OBJPROP_PERIOD,period);
//--- установим масштаб
ObjectSetInteger(chart_ID,name,OBJPROP_CHART_SCALE,scale);
//--- отобразим (true) или скроем (false) шкалу времени
ObjectSetInteger(chart_ID,name,OBJPROP_DATE_SCALE,date_scale);
//--- отобразим (true) или скроем (false) шкалу цены
ObjectSetInteger(chart_ID,name,OBJPROP_PRICE_SCALE,price_scale);
//--- установим цвет рамки при включенном режиме выделения объекта
ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии рамки при включенном режиме выделения объекта
ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер точки привязки, с помощью которой можно перемещать объект
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Устанавливает символ и таймфрейм объекта "График" |
//+-----+
bool ObjectChartSetSymbolAndPeriod(const long          chart_ID=0,      // ID графика
                                    const string        name="Chart",    // имя объекта
                                    const string        symbol="EURUSD", // символ
                                    const ENUM_TIMEFRAMES period=PERIOD_H1) // таймфрейм
{
//--- сбросим значение ошибки
ResetLastError();
//--- установим символ и таймфрейм объекта "График"
if(!ObjectSetString(chart_ID,name,OBJPROP_SYMBOL,symbol))
{
Print(__FUNCTION__,
": не удалось установить символ для объекта \"График\"! Код ошибки = ",GetLastError());
return(false);
}
if(!ObjectSetInteger(chart_ID,name,OBJPROP_PERIOD,period))
{
Print(__FUNCTION__,
": не удалось установить период для объекта \"График\"! Код ошибки = ",GetLastError());
return(false);
}
}

```

```

        }

//--- успешное выполнение
    return(true);
}

//+-----+
//| Изменяет размер объекта "График" |
//+-----+

bool ObjectChartChangeSize(const long    chart_ID=0,      // ID графика (не объекта)
                           const string name="Chart", // имя объекта
                           const int      width=300,   // ширина
                           const int      height=200) // высота

{
    //--- сбросим значение ошибки
    ResetLastError();

    //--- изменим размеры объекта
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
    {
        Print(__FUNCTION__,
              ": не удалось изменить ширину объекта \"График\"! Код ошибки = ",
              GetLastError());
        return(false);
    }

    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
    {
        Print(__FUNCTION__,
              ": не удалось изменить высоту объекта \"График\"! Код ошибки = ",
              GetLastError());
        return(false);
    }

    //--- успешное выполнение
    return(true);
}

//+-----+
//| Изменяет размер объекта "График" |
//+-----+

```

```

Print(__FUNCTION__,
      ": не удалось изменить высоту объекта \"График\"! Код ошибки = ", GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Возвращает ID объекта "График" |
//+-----+
long ObjectChartGetID(const long    chart_ID=0,      // ID графика (не объекта)
                      const string name="Chart") // имя объекта
{
//--- подготовим переменную для получения ID объекта "График"
long id=-1;
//--- сбросим значение ошибки
ResetLastError();
//--- получим ID
if(!ObjectGetInteger(chart_ID,name,OBJPROP_CHART_ID,0,id))
{
    Print(__FUNCTION__,
          ": не удалось получить ID объекта \"График\"! Код ошибки = ", GetLastError());
}
//--- возврат результата
return(id);
}

//+-----+
//| Удаляет объект "График" |
//+-----+
bool ObjectChartDelete(const long    chart_ID=0,      // ID графика (не объекта)
                      const string name="Chart") // имя объекта
{
//--- сбросим значение ошибки
ResetLastError();
//--- удалим кнопку
if(!ObjectDelete(chart_ID,name))
{
    Print(__FUNCTION__,
          ": не удалось удалить объект \"График\"! Код ошибки = ", GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
}

```

```

//--- получим количество символов в "Обзоре рынка"
int symbols=SymbolsTotal(true);
//--- определим, есть ли символ с указанным именем в списке символов
bool exist=false;
for(int i=0;i<symbols;i++)
{
    if(InpSymbol==SymbolName(i,true))
    {
        exist=true;
        break;
    }
}
if(!exist)
{
    Print("Ошибка! Данный символ ",InpSymbol," не представлен в окне \"Обзор Рынка\"");
    return;
}

//--- проверка входных параметров на корректность
if(InpScale<0 || InpScale>5)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}

//--- размеры окна графика
long x_distance;
long y_distance;
//--- определим размеры окна
if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
{
    Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
    return;
}
if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
{
    Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
    return;
}

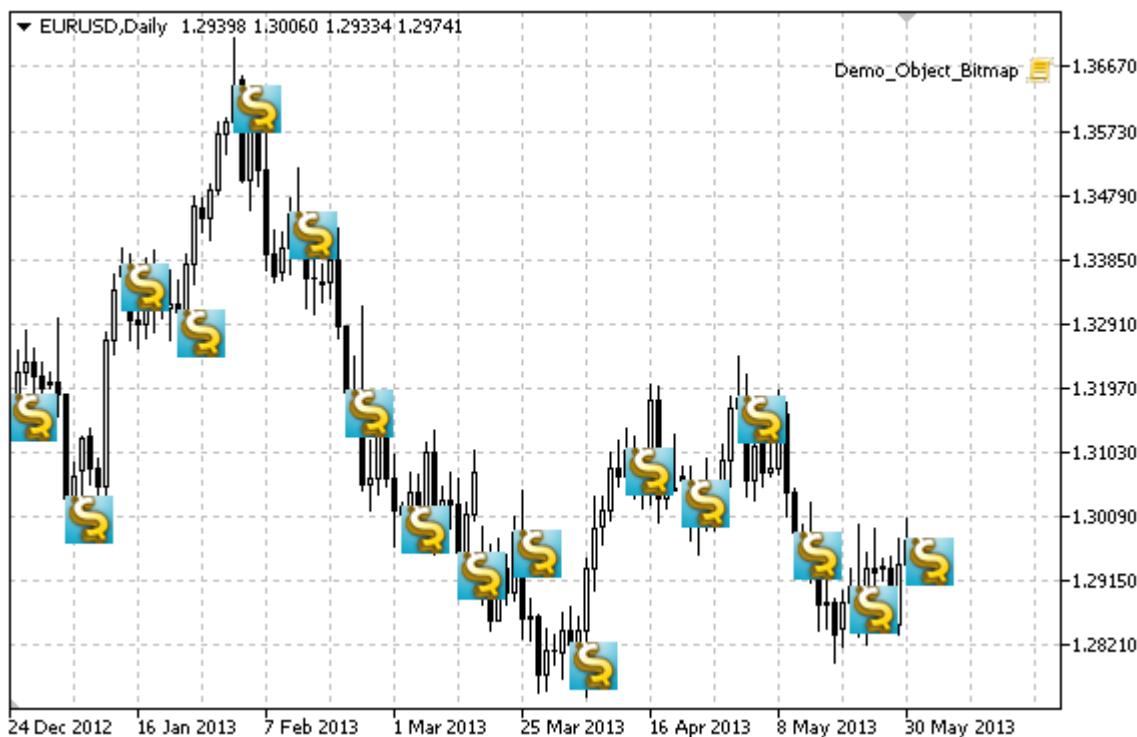
//--- установим координаты объекта "График" и его размер
int x=(int)x_distance/16;
int y=(int)y_distance/16;
int x_size=(int)x_distance*7/16;
int y_size=(int)y_distance*7/16;
//--- создадим объект "График"
if(!ObjectChartCreate(0,InpName,0,InpSymbol,InpPeriod,x,y,x_size,y_size,InpCorner,:
    InpPriceScale,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidden,Inp
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();

```

```
Sleep(1000);
//--- растянем объект "График"
int steps=(int)MathMin(x_distance*7/16,y_distance*7/16);
for(int i=0;i<steps;i++)
{
    //--- изменим размеры
    x_size+=1;
    y_size+=1;
    if(!ObjectChartChangeSize(0,InpName,x_size,y_size))
        return;
    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;
    //--- перерисуем график и подождем 0.01 секунды
    ChartRedraw();
    Sleep(10);
}
//--- задержка в полсекунды
Sleep(500);
//--- изменим таймфрейм графика
if(!ObjectChartSetSymbolAndPeriod(0,InpName,InpSymbol,PERIOD_M1))
    return;
ChartRedraw();
//--- задержка в три секунды
Sleep(3000);
//--- удалим объект
ObjectChartDelete(0,InpName);
ChartRedraw();
//--- подождем 1 секунду
Sleep(1000);
//---
```

## OBJ\_BITMAP

Объект "Рисунок".



### Примечание

Для объекта "Рисунок" можно выбирать [область видимости](#) рисунка.

### Пример

Следующий скрипт создает на графике несколько картинок. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт создает рисунок в окне графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpFile="\Images\dollar.bmp"; // Имя файла с картинкой
input int                InpWidth=24;                  // X-координата области видимости
input int                InpHeight=24;                 // Y-координата области видимости
input int                InpXOffset=4;                 // Смещение области видимости по X
input int                InpYOffset=4;                 // Смещение области видимости по Y
input color              InpColor=clrRed;               // Цвет рамки при выделении
input ENUM_LINE_STYLE    InpStyle=STYLE_SOLID;        // Стиль линии при выделении
input int                InpPointSize=1;                // Размер точки для перемещений
input bool               InpBack=false;                // Объект на заднем плане
input bool               InpSelection=false;           // Выделить для перемещений

```

```

input bool           InpHidden=true;           // Скрыт в списке объектов
input long          InpZOrder=0;             // Приоритет на нажатие мышью
//+-----+
//| Создает рисунок в окне графика
//+-----+
bool BitmapCreate(const long           chart_ID=0,          // ID графика
                   const string        name="Bitmap",       // имя рисунка
                   const int            sub_window=0,      // номер подокна
                   datetime            time=0,           // время точки привязки
                   double               price=0,          // цена точки привязки
                   const string         file="",          // имя файла с картинкой
                   const int            width=10,         // X-координата области видимости
                   const int            height=10,        // Y-координата области видимости
                   const int            x_offset=0,       // смещение области видимости
                   const int            y_offset=0,       // смещение области видимости
                   const color          clr=clrRed,       // цвет рамки при выделении
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии при выделении
                   const int            point_width=1,    // размер точки перемещения
                   const bool            back=false,        // на заднем плане
                   const bool            selection=false,   // выделить для перемещения
                   const bool            hidden=true,       // скрыт в списке объектов
                   const long            z_order=0)        // приоритет на нажатие мышью
{
//--- установим координаты точки привязки, если они не заданы
    ChangeBitmapEmptyPoint(time,price);
//---бросим значение ошибки
    ResetLastError();
//--- создадим рисунок
    if(!ObjectCreate(chart_ID,name,OBJ_BITMAP,sub_window,time,price))
    {
        Print(__FUNCTION__,
              ": не удалось создать рисунок в окне графика! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим путь к файлу с картинкой
    if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
    {
        Print(__FUNCTION__,
              ": не удалось загрузить картинку! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим область видимости изображения; если значения ширины или высоты
//--- больше значений ширины и высоты (соответственно) исходного изображения, то
//--- оно не рисуется; если значения ширины и высоты меньше размеров изображения,
//--- то рисуется та его часть, которая соответствует этим размерам
    ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
    ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим, какая часть изображения должна показываться в области видимости
//--- по умолчанию это левая верхняя область изображения; значения позволяют

```

```

//--- произвести сдвиг от этого угла и отобразить другую часть изображения
ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);

//--- установим цвет рамки при включенном режиме выделения объекта
ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);

//--- установим стиль линии рамки при включенном режиме выделения объекта
ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);

//--- установим размер точки привязки, с помощью которой можно перемещать объект
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);

//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);

//--- включим (true) или отключим (false) режим перемещения метки мышью
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

//--- успешное выполнение
return(true);
}

//+-----+
//| Устанавливает новую картинку для рисунка |
//+-----+
bool BitmapSetImage(const long chart_ID=0, // ID графика
                     const string name="Bitmap", // имя рисунка
                     const string file="") // путь к файлу
{
//--- сбросим значение ошибки
ResetLastError();

//--- установим путь к файлу с картинкой
if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
{
Print(__FUNCTION__,
": не удалось загрузить картинку! Код ошибки = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает рисунок в окне графика |
//+-----+
bool BitmapMove(const long chart_ID=0, // ID графика
                const string name="Bitmap", // имя рисунка
                datetime time=0, // время точки привязки
                double price=0) // цена точки привязки
{
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
}

```

```

if(!time)
    time=TimeCurrent();
if(!price)
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
ResetLastError();
//--- переместим точку привязки
if(!ObjectMove(chart_ID,name,0,time,price))
{
    Print(__FUNCTION__,
        ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет размер области видимости (размер рисунка) |
//+-----+
bool BitmapChangeSize(const long    chart_ID=0,      // ID графика
                      const string name="Bitmap", // имя рисунка
                      const int     width=0,       // ширина рисунка
                      const int     height=0)     // высота рисунка
{
//--- сбросим значение ошибки
ResetLastError();
//--- изменим размеры рисунка
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
{
    Print(__FUNCTION__,
        ": не удалось изменить ширину рисунка! Код ошибки = ",GetLastError());
    return(false);
}
if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
{
    Print(__FUNCTION__,
        ": не удалось изменить высоту рисунка! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет координату левого верхнего угла области видимости |
//+-----+
bool BitmapMoveVisibleArea(const long    chart_ID=0,      // ID графика
                           const string name="Bitmap", // имя рисунка
                           const int     x_offset=0,    // координата X области видимости
                           const int     y_offset=0)    // координата Y области видимости

```

```

{
//--- сбросим значение ошибки
ResetLastError();

//--- изменим координаты области видимости рисунка
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
{
Print(__FUNCTION__,
": не удалось изменить X-координату области видимости! Код ошибки = ",GetLastError());
return(false);
}

if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
{
Print(__FUNCTION__,
": не удалось изменить Y-координату области видимости! Код ошибки = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

//-----+
//| Удаляет рисунок |
//-----+
bool BitmapDelete(const long chart_ID=0, // ID графика
                  const string name="Bitmap") // имя рисунка
{
//--- сбросим значение ошибки
ResetLastError();

//--- удалим метку
if(!ObjectDelete(chart_ID,name))
{
Print(__FUNCTION__,
": не удалось удалить рисунок! Код ошибки = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

//-----+
//| Проверяет значения точки привязки и для пустых значений |
//| устанавливает значения по умолчанию |
//-----+
void ChangeBitmapEmptyPoint(datetime &time,double &price)
{
//--- если время точки не задано, то она будет на текущем баре
if(!time)
time=TimeCurrent();

//--- если цена точки не задана, то она будет иметь значение Bid
if(!price)
price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
}

```

```

    }

//+-----+
//| Script program start function | 
//+-----+

void OnStart()
{
    datetime date[]; // массив для хранения дат видимых баров
    double   close[]; // массив для хранения цен Close

//--- имя файла с картинкой
    string   file="\Images\dollar.bmp";
//--- количество видимых баров в окне графика
    int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- выделение памяти
    ArrayResize(date,bars);
    ArrayResize(close,bars);
//--- заполним массив дат
    ResetLastError();
    if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
    {
        Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
        return;
    }
//--- заполним массив цен Close
    if(CopyClose(Symbol(),Period(),0,bars,close)==-1)
    {
        Print("Не удалось скопировать значения цен Close! Код ошибки = ",GetLastError());
        return;
    }
//--- определим, как часто надо выводить картинки
    int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- определим шаг
    int step=1;
    switch(scale)
    {
        case 0:
            step=27;
            break;
        case 1:
            step=14;
            break;
        case 2:
            step=7;
            break;
        case 3:
            step=4;
            break;
        case 4:
            step=2;
            break;
    }
}

```

```
        }

//--- создадим рисунки для значений High и Low баров (с промежутками)
for(int i=0;i<bars;i+=step)
{
    //--- создаем рисунки
    if(!BitmapCreate(0,"Bitmap_"+(string)i,0,date[i],close[i],InpFile,InpWidth,InpHeight,
                    InpYOffset,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidden,InpTransparency))
    {
        return;
    }

    //--- проверим факт принудительного завершения скрипта
    if(IsStopped())
        return;

    //--- перерисуем график
    ChartRedraw();

    // задержка в 0.05 секунды
    Sleep(50);

}

//--- задержка в полсекунды
Sleep(500);

//--- удалим значки "Sell"
for(int i=0;i<bars;i+=step)
{
    if(!BitmapDelete(0,"Bitmap_"+(string)i))
        return;
    if(!BitmapDelete(0,"Bitmap_"+(string)i))
        return;

    //--- перерисуем график
    ChartRedraw();

    // задержка в 0.05 секунды
    Sleep(50);
}

//---
```

## OBJ\_BITMAP\_LABEL

Объект "Графическая метка".



### Примечание

Положение точки привязки относительно метки можно выбрать из перечисления [ENUM\\_ANCHOR\\_POINT](#). Координаты точки привязки задаются в пикселях.

Также можно выбрать угол привязки графической метки из перечисления [ENUM\\_BASE\\_CORNER](#).

Для объекта "Графическая метка" можно выбирать [область видимости](#) рисунка.

### Пример

Следующий скрипт создает на графике несколько картинок. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт создает объект \"Графическая метка\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="BmpLabel";           // Имя метки
input string           InpFileOn="\Images\dollar.bmp"; // Имя файла для режима On
input string           InpFileOff="\Images\euro.bmp"; // Имя файла для режима Off
input bool             InpState=false;              // Метка нажата/отжата
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Угол графика для привязки
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_CENTER;    // Способ привязки
input color            InpColor=clrRed;              // Цвет рамки при выделении

```

```

input ENUM_LINE_STYLE    InpStyle=STYLE_SOLID;           // Стиль линии при выделении
input int                 InpPointSize=1;                // Размер точки для перемещения
input bool                InpBack=false;                // Объект на заднем плане
input bool                InpSelection=false;            // Выделить для перемещений
input bool                InpHidden=true;                // Скрыт в списке объектов
input long                InpZOrder=0;                  // Приоритет на нажатие мыши
//+-----+
//| Создает объект "Графическая метка" |
//+-----+
bool BitmapLabelCreate(const long          chart_ID=0,           // ID графика
                       const string        name="BmpLabel",       // имя метки
                       const int           sub_window=0,      // номер подокна
                       const int           x=0,              // координата X
                       const int           y=0,              // координата Y
                       const string        file_on="",        // картинка в режиме On
                       const string        file_off="",       // картинка в режиме Off
                       const int           width=0,          // X-координата
                       const int           height=0,         // Y-координата
                       const int           x_offset=10,       // смещение от X
                       const int           y_offset=10,       // смещение от Y
                       const bool          state=false,       // нажата/отжата
                       const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // угол графика
                       const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // способ прикрепления
                       const color          clr=clrRed,        // цвет рамки
                       const ENUM_LINE_STYLE style=STYLE_SOLID,     // стиль линии
                       const int           point_width=1,       // размер точки
                       const bool          back=false,         // на заднем плане
                       const bool          selection=false,     // выделить для перемещения
                       const bool          hidden=true,        // скрыт в списке
                       const long           z_order=0)        // приоритет
{
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим графическую метку
    if(!ObjectCreate(chart_ID, name, OBJ_BITMAP_LABEL, sub_window, 0, 0))
    {
        Print(__FUNCTION__, 
              ": не удалось создать объект \"Графическая метка\"! Код ошибки = ", GetLastError());
        return(false);
    }
//--- установим картинки для режимов On и Off
    if(!ObjectSetString(chart_ID, name, OBJPROP_BMPFILE, 0, file_on))
    {
        Print(__FUNCTION__, 
              ": не удалось загрузить картинку для режима On! Код ошибки = ", GetLastError());
        return(false);
    }
    if(!ObjectSetString(chart_ID, name, OBJPROP_BMPFILE, 1, file_off))
    {

```

```

Print(__FUNCTION__,
      ": не удалось загрузить картинку для режима Off! Код ошибки = ",GetLastError());
return(false);
}

//--- установим координаты метки
ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);

//--- установим область видимости изображения; если значения ширины или высоты
//--- больше значений ширины и высоты (соответственно) исходного изображения, то
//--- оно не рисуется; если значения ширины и высоты меньше размеров изображения,
//--- то рисуется та его часть, которая соответствует этим размерам
ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);

//--- установим, какая часть изображения должна показываться в области видимости
//--- по умолчанию это левая верхняя область изображения; значения позволяют
//--- произвести сдвиг от этого угла и отобразить другую часть изображения
ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);

//--- установим, в каком состоянии находится метка (нажатом или отжатом)
ObjectSetInteger(chart_ID,name,OBJPROP_STATE,state);

//--- установим угол графика, относительно которого будут определяться координаты точки
ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);

//--- установим способ привязки
ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);

//--- установим цвет рамки при включенном режиме выделения объекта
ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);

//--- установим стиль линии рамки при включенном режиме выделения объекта
ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);

//--- установим размер точки привязки, с помощью которой можно перемещать объект
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);

//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);

//--- включим (true) или отключим (false) режим перемещения метки мышью
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);

//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);

//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);

//--- успешное выполнение
return(true);
}

//-----+
//| Устанавливает новую картинку для объекта "Графическая метка" |
//-----+
bool BitmapLabelSetImage(const long    chart_ID=0,           // ID графика
                        const string name="BmpLabel", // имя метки
                        const int     on_off=0,        // модификатор (On или Off)
                        const string file="")        // путь к файлу

```

```

{
//--- сбросим значение ошибки
ResetLastError();

//--- установим путь к файлу с картинкой
if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,on_off,file))
{
Print(__FUNCTION__,
": не удалось загрузить картинку! Код ошибки = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает объект "Графическая метка" |
//+-----+
bool BitmapLabelMove(const long    chart_ID=0,          // ID графика
                     const string name="BmpLabel", // имя метки
                     const int      x=0,           // координата по оси X
                     const int      y=0)          // координата по оси Y
{
//--- сбросим значение ошибки
ResetLastError();

//--- переместим объект
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
{
Print(__FUNCTION__,
": не удалось переместить X-координату объекта! Код ошибки = ",GetLastError());
return(false);
}

if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
{
Print(__FUNCTION__,
": не удалось переместить Y-координату объекта! Код ошибки = ",GetLastError());
return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет размер области видимости (размер объекта) |
//+-----+
bool BitmapLabelChangeSize(const long    chart_ID=0,          // ID графика
                           const string name="BmpLabel", // имя метки
                           const int      width=0,        // ширина метки
                           const int      height=0)       // высота метки
{
//--- сбросим значение ошибки
ResetLastError();
}

```

```

//--- изменим размеры объекта
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
{
    Print(__FUNCTION__,
        ": не удалось изменить ширину объекта! Код ошибки = ",GetLastError());
    return(false);
}
if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
{
    Print(__FUNCTION__,
        ": не удалось изменить высоту объекта! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
// | Изменяет координату левого верхнего угла области видимости |
//+-----+
bool BitmapLabelMoveVisibleArea(const long    chart_ID=0,          // ID графика
                                 const string name="BmpLabel", // имя метки
                                 const int      x_offset=0,     // координата X области
                                 const int      y_offset=0)     // координата Y области
{
//--- сбросим значение ошибки
ResetLastError();

//--- изменим координаты области видимости объекта
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
{
    Print(__FUNCTION__,
        ": не удалось изменить X-координату области видимости! Код ошибки = ",GetLastError());
    return(false);
}
if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
{
    Print(__FUNCTION__,
        ": не удалось изменить Y-координату области видимости! Код ошибки = ",GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
// | Удаляет объект "Графическая метка" |
//+-----+
bool BitmapLabelDelete(const long    chart_ID=0,          // ID графика
                      const string name="BmpLabel") // имя метки
{
//--- сбросим значение ошибки

```

```

        ResetLastError();

//--- удалим метку
if(!ObjectDelete(chart_ID, name))
{
    Print(__FUNCTION__,
          ": не удалось удалить объект \"Графическая метка\"! Код ошибки = ", GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

//-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- размеры окна графика
long x_distance;
long y_distance;
//--- определим размеры окна
if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
{
    Print("Не удалось получить ширину графика! Код ошибки = ", GetLastError());
    return;
}
if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
{
    Print("Не удалось получить высоту графика! Код ошибки = ", GetLastError());
    return;
}

//--- определим координаты графической метки
int x=(int)x_distance/2;
int y=(int)y_distance/2;
//--- установим размеры метки и координаты области видимости
int width=32;
int height=32;
int x_offset=0;
int y_offset=0;
//--- разместим графическую метку по центру окна
if(!BitmapLabelCreate(0,InpName,0,x,y,InpFileOn,InpFileOff,width,height,x_offset,y_offset,
                      InpCorner,InpAnchor,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHide))
{
    return;
}

//--- перерисуем график и подождем одну секунду
ChartRedraw();
Sleep(1000);
//--- изменим размеры области видимости метки в цикле
for(int i=0;i<6;i++)

```

```
{  
    //--- изменим размеры области видимости  
    width--;  
    height--;  
    if(!BitmapLabelChangeSize(0, InpName, width, height))  
        return;  
    //--- проверим факт принудительного завершения скрипта  
    if(IsStopped())  
        return;  
    //--- перерисуем график  
    ChartRedraw();  
    // задержка в 0.3 секунды  
    Sleep(300);  
}  
//--- задержка в 1 секунду  
Sleep(1000);  
//--- изменим координаты области видимости метки в цикле  
for(int i=0;i<2;i++)  
{  
    //--- изменим координаты области видимости  
    x_offset++;  
    y_offset++;  
    if(!BitmapLabelMoveVisibleArea(0, InpName, x_offset, y_offset))  
        return;  
    //--- проверим факт принудительного завершения скрипта  
    if(IsStopped())  
        return;  
    //--- перерисуем график  
    ChartRedraw();  
    // задержка в 0.3 секунды  
    Sleep(300);  
}  
//--- задержка в 1 секунду  
Sleep(1000);  
//--- удалим метку  
BitmapLabelDelete(0, InpName);  
ChartRedraw();  
//--- задержка в 1 секунду  
Sleep(1000);  
//---  
}
```

## OBJ\_EDIT

Объект "Поле ввода".



### Примечание

Координаты точки привязки задаются в пикселях. Можно выбрать угол привязки "Поля ввода" из перечисления [ENUM\\_BASE\\_CORNER](#).

Также можно выбрать один из типов выравнивания текста внутри "Поля ввода" из перечисления [ENUM\\_ALIGN\\_MODE](#).

### Пример

Следующий скрипт создает и перемещает на графике объект "Поле ввода". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает объект \"Поле ввода\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="Edit";           // Имя объекта
input string          InpText="Text";            // Текст объекта
input string          InpFont="Arial";           // Шрифт
input int             InpFontSize=14;           // Размер шрифта
input ENUM_ALIGN_MODE InpAlign=ALIGN_CENTER;    // Способ выравнивания текста
input bool             InpReadOnly=false;         // Возможность редактировать
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Угол графика для привязки
input color            InpColor=clrBlack;          // Цвет текста
```

```

input color           InpBackColor=clrWhite;          // Цвет фона
input color           InpBorderColor=clrBlack;         // Цвет границы
input bool            InpBack=false;                 // Объект на заднем плане
input bool            InpSelection=false;             // Выделить для перемещений
input bool            InpHidden=true;                // Скрыт в списке объектов
input long             InpZOrder=0;                  // Приоритет на нажатие мышью
//+-----+
//| Создает объект "Поле ввода"
//+-----+
bool EditCreate(const long           chart_ID=0,           // ID графика
                const string        name="Edit",           // имя объекта
                const int            sub_window=0,         // номер подокна
                const int            x=0,                  // координата по оси
                const int            y=0,                  // координата по оси
                const int            width=50,             // ширина
                const int            height=18,            // высота
                const string         text="Text",           // текст
                const string         font="Arial",          // шрифт
                const int            font_size=10,          // размер шрифта
                const ENUM_ALIGN_MODE align=ALIGN_CENTER, // способ выравнивани
                const bool            read_only=false,       // возможность редакт
                const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // угол графика для и
                const color           clr=clrBlack,          // цвет текста
                const color           back_clr=clrWhite,        // цвет фона
                const color           border_clr=clrNONE,       // цвет границы
                const bool            back=false,            // на заднем плане
                const bool            selection=false,        // выделить для перем
                const bool            hidden=true,            // скрыт в списке обь
                const long             z_order=0)            // приоритет на нажат
{
//--- сбросим значение ошибки
    ResetLastError();
//--- создадим поле ввода
    if(!ObjectCreate(chart_ID,name,OBJ_EDIT,sub_window,0,0))
    {
        Print(__FUNCTION__,
              ": не удалось создать объект \"Поле ввода\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- установим координаты объекта
    ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
    ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- установим размеры объекта
    ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
    ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим текст
    ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- установим шрифт текста
    ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
}

```

```

//--- установим размер шрифта
ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- установим способ выравнивания текста в объекте
ObjectSetInteger(chart_ID,name,OBJPROP_ALIGN,align);
//--- установим (true) или отменим (false) режим только для чтения
ObjectSetInteger(chart_ID,name,OBJPROP_READONLY,readonly);
//--- установим угол графика, относительно которого будут определяться координаты объекта
ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- установим цвет текста
ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим цвет фона
ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- установим цвет границы
ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает объект "Поле ввода" |
//+-----+
bool EditMove(const long    chart_ID=0, // ID графика
              const string name="Edit", // имя объекта
              const int      x=0,        // координата по оси X
              const int      y=0)        // координата по оси Y
{
//--- сбросим значение ошибки
ResetLastError();
//--- переместим объект
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
{
Print(__FUNCTION__,
": не удалось переместить X-координату объекта! Код ошибки = ",GetLastError());
return(false);
}
if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
{
Print(__FUNCTION__,
": не удалось переместить Y-координату объекта! Код ошибки = ",GetLastError());
return(false);
}
}

```

```

//--- успешное выполнение
    return(true);
}
//+-----+
//| Изменяет размеры объекта "Поле ввода" |
//+-----+
bool EditChangeSize(const long    chart_ID=0, // ID графика
                     const string name="Edit", // имя объекта
                     const int      width=0,   // ширина
                     const int      height=0) // высота
{
//--- сбросим значение ошибки
    ResetLastError();
//--- изменим размеры объекта
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
    {
        Print(__FUNCTION__,
              ": не удалось изменить ширину объекта! Код ошибки = ",GetLastError());
        return(false);
    }
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
    {
        Print(__FUNCTION__,
              ": не удалось изменить высоту объекта! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Изменяет текст объекта "Поле ввода" |
//+-----+
bool EditTextChange(const long    chart_ID=0, // ID графика
                     const string name="Edit", // имя объекта
                     const string text="Text") // текст
{
//--- сбросим значение ошибки
    ResetLastError();
//--- изменим текст объекта
    if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
    {
        Print(__FUNCTION__,
              ": не удалось изменить текст! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+

```

```

//| Возвращает текст объекта "Поле ввода"
//+-----+
bool EditTextGet(string      &text,          // текст
                  const long   chart_ID=0,    // ID графика
                  const string  name="Edit") // имя объекта
{
//--- сбросим значение ошибки
    ResetLastError();
//--- получим текст объекта
    if(!ObjectGetString(chart_ID,name,OBJPROP_TEXT,0,text))
    {
        Print(__FUNCTION__,
              ": не удалось получить текст! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Удаляет объект "Поле ввода"
//+-----+
bool EditDelete(const long   chart_ID=0,    // ID графика
                 const string  name="Edit") // имя объекта
{
//--- сбросим значение ошибки
    ResetLastError();
//--- удалим метку
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить объект \"Поле ввода\"! Код ошибки = ",GetLastError());
        return(false);
    }
//--- успешное выполнение
    return(true);
}
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- размеры окна графика
    long x_distance;
    long y_distance;
//--- определим размеры окна
    if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
    {
        Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
        return;
    }
}

```

```

    }

    if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
    {
        Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
        return;
    }

    //--- определим шаг для изменения размеров поля ввода
    int x_step=(int)x_distance/64;

    //--- установим координаты поля ввода и его размер
    int x=(int)x_distance/8;
    int y=(int)y_distance/2;
    int x_size=(int)x_distance/8;
    int y_size=InpFontSize*2;

    //--- запомним текст в локальную переменную
    string text=InpText;

    //--- создадим поле ввода
    if(!EditCreate(0,InpName,0,x,y,x_size,y_size,InpText,InpFont,InpFontSize,InpAlign,InpCorner,InpColor,InpBackColor,InpBorderColor,InpBack,InpSelection,InpHidden,InpText))
    {
        return;
    }

    //--- перерисуем график и подождем 1 секунду
    ChartRedraw();
    Sleep(1000);

    //--- растянем поле ввода
    while(x_size-x<x_distance*5/8)
    {
        //--- увеличим ширину поля ввода
        x_size+=x_step;
        if(!EditChangeSize(0,InpName,x_size,y_size))
            return;

        //--- проверим факт принудительного завершения скрипта
        if(IsStopped())
            return;

        //--- перерисуем график и подождем 0.05 секунды
        ChartRedraw();
        Sleep(50);
    }

    //--- задержка в полсекунды
    Sleep(500);

    //--- изменим текст
    for(int i=0;i<20;i++)
    {
        //--- прибавим "+" в начале и конце
        text)+"+"+text+"+";
        if(!EditTextChange(0,InpName,text))
            return;

        //--- проверим факт принудительного завершения скрипта
        if(IsStopped())
    }
}

```

```
    return;
//--- перерисуем график и подождем 0.1 секунды
ChartRedraw();
Sleep(100);
}

//--- задержка в полсекунды
Sleep(500);
//--- удалим поле ввода
EditDelete(0, InpName);
ChartRedraw();
//--- подождем 1 секунду
Sleep(1000);
//---
```

## OBJ\_EVENT

Объект "Событие".



### Примечание

При наведении на событие курсора мыши, всплывает его текст.

### Пример

Следующий скрипт создает и перемещает на графике объект "Событие". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт строит графический объект \"Событие\"."
#property description "Дата точки привязки задается в процентах от"
#property description "ширины окна графика в барах."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="Event";      // Имя события
input int         InpDate=25;          // Дата события в %
input string      InpText="Text";       // Текст события
input color        InpColor=clrRed;     // Цвет события
input int         InpWidth=1;          // Размер точки при выделении
input bool         InpBack=false;       // Событие на заднем плане
input bool         InpSelection=false;  // Выделить для перемещений
input bool         InpHidden=true;      // Скрыт в списке объектов

```

```

input long           InpZOrder=0;          // Приоритет на нажатие мышью
//+-----+
//| Создает объект "Событие" на графике
//+-----+
bool EventCreate(const long           chart_ID=0,        // ID графика
                  const string      name="Event",     // имя события
                  const int         sub_window=0,    // номер подокна
                  const string      text="Text",      // текст события
                  datetime         time=0,         // время
                  const color       clr=clrRed,     // цвет
                  const int         width=1,        // толщина точки при выделении
                  const bool        back=false,     // на заднем плане
                  const bool        selection=false, // выделить для перемещений
                  const bool        hidden=true,    // скрыт в списке объектов
                  const long         z_order=0)     // приоритет на нажатие мышью
{
//--- если время не задано, то создаем объект на последнем баре
  if(!time)
    time=TimeCurrent();
//--- сбросим значение ошибки
  ResetLastError();
//--- создадим объект "Событие"
  if(!ObjectCreate(chart_ID, name, OBJ_EVENT, sub_window, time, 0))
  {
    Print(__FUNCTION__,
          ": не удалось создать объект \"Событие\"! Код ошибки = ", GetLastError());
    return(false);
  }
//--- установим текст события
  ObjectSetString(chart_ID, name, OBJPROP_TEXT, text);
//--- установим цвет
  ObjectSetInteger(chart_ID, name, OBJPROP_COLOR, clr);
//--- установим толщину точки привязки, если объект выделен
  ObjectSetInteger(chart_ID, name, OBJPROP_WIDTH, width);
//--- отобразим на переднем (false) или заднем (true) плане
  ObjectSetInteger(chart_ID, name, OBJPROP_BACK, back);
//--- включим (true) или отключим (false) режим перемещения события мышью
  ObjectSetInteger(chart_ID, name, OBJPROP_SELECTABLE, selection);
  ObjectSetInteger(chart_ID, name, OBJPROP_SELECTED, selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
  ObjectSetInteger(chart_ID, name, OBJPROP_HIDDEN, hidden);
//--- установим приоритет на получение события нажатия мыши на графике
  ObjectSetInteger(chart_ID, name, OBJPROP_ZORDER, z_order);
//--- успешное выполнение
  return(true);
}
//+-----+
//| Изменяет текст объекта "Событие"
//+-----+

```

```

bool EventTextChange(const long chart_ID=0, // ID графика
                     const string name="Event", // имя события
                     const string text="Text") // текст
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- изменим текст объекта
    if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
    {
        Print(__FUNCTION__,
              ": не удалось изменить текст! Код ошибки = ",GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

//+-----+
//| Перемещение объекта "Событие" |
//+-----+
bool EventMove(const long chart_ID=0, // ID графика
               const string name="Event", // имя события
               datetime time=0) // время
{
    //--- если время не задано, то перемещаем событие на последний бар
    if(!time)
        time=TimeCurrent();
    //--- сбросим значение ошибки
    ResetLastError();
    //--- переместим объект
    if(!ObjectMove(chart_ID,name,0,time,0))
    {
        Print(__FUNCTION__,
              ": не удалось переместить объект \"Событие\"! Код ошибки = ",GetLastError());
        return(false);
    }
    //--- успешное выполнение
    return(true);
}

//+-----+
//| Удаляет объект "Событие" |
//+-----+
bool EventDelete(const long chart_ID=0, // ID графика
                 const string name="Event") // имя события
{
    //--- сбросим значение ошибки
    ResetLastError();
    //--- удалим объект
    if(!ObjectDelete(chart_ID,name))
    {
}

```

```

Print(__FUNCTION__,
      ": не удалось удалить объект \"Событие\"! Код ошибки = ", GetLastError());
return(false);
}
//--- успешное выполнение
return(true);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- проверим входные параметры на корректность
if(InpDate<0 || InpDate>100)
{
    Print("Ошибка! Некорректные значения входных параметров!");
    return;
}
//--- количество видимых баров в окне графика
int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- массив для хранения значений дат, которые будут использованы
//--- для установки и изменения координаты точки привязки объекта "Событие"
datetime date[];
//--- выделение памяти
ArrayResize(date,bars);
//--- заполним массив дат
ResetLastError();
if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
{
    Print("Не удалось скопировать значения времени! Код ошибки = ", GetLastError());
    return;
}
//--- определим точки для создания объекта
int d=InpDate*(bars-1)/100;
//--- создадим объект "Событие"
if(!EventCreate(0,InpName,0,InpText,date[d],InpColor,InpWidth,
InpBack,InpSelection,InpHidden,InpZOrder))
{
    return;
}
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);
//--- теперь будем перемещать объект
//--- счетчик цикла
int h_steps=bars/2;
//--- перемещаем объект
for(int i=0;i<h_steps;i++)
{
}

```

```
//--- возьмем следующее значение
if(d<bars-1)
    d+=1;
//--- сдвигаем точку
if(!EventMove(0, InpName, date[d]))
    return;
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график
ChartRedraw();
// задержка в 0.05 секунды
Sleep(50);
}
//--- задержка в 1 секунду
Sleep(1000);
//--- удалим канал с графика
EventDelete(0, InpName);
ChartRedraw();
//--- задержка в 1 секунду
Sleep(1000);
//---
```

## OBJ\_RECTANGLE\_LABEL

Объект "Прямоугольная метка".



### Примечание

Координаты точки привязки задаются в пикселях. Можно выбрать угол привязки прямоугольной метки из перечисления [ENUM\\_BASE\\_CORNER](#). Тип рамки для прямоугольной метки можно выбрать из перечисления [ENUM\\_BORDER\\_TYPE](#).

Предназначена для создания и оформления пользовательского графического интерфейса.

### Пример

Следующий скрипт создает и перемещает на графике объект "Прямоугольная метка". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```

//--- описание
#property description "Скрипт создает графический объект \"Прямоугольная метка\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string      InpName="RectLabel";           // Имя метки
input color       InpBackColor=clrSkyBlue;        // Цвет фона
input ENUM_BORDER_TYPE InpBorder=BORDER_FLAT;    // Тип рамки
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Угол графика для привязки
input color       InpColor=clrDarkBlue;           // Цвет плоской границы (Flat)
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID;        // Стиль плоской границы (Flat)
input int         InpLineWidth=3;                 // Толщина плоской границы (Flat)

```

```

input bool           InpBack=false;           // Объект на заднем плане
input bool           InpSelection=true;        // Выделить для перемещений
input bool           InpHidden=true;          // Скрыт в списке объектов
input long           InpZOrder=0;             // Приоритет на нажатие мышью
//+-----+
//| Создает прямоугольную метку
//+-----+
bool RectLabelCreate(const long           chart_ID=0,           // ID графика
                      const string         name="RectLabel",      // имя метки
                      const int            sub_window=0,       // номер подокна
                      const int            x=0,                 // координата по x
                      const int            y=0,                 // координата по y
                      const int            width=50,            // ширина
                      const int            height=18,           // высота
                      const color          back_clr=C'236,233,216', // цвет фона
                      const ENUM_BORDER_TYPE border=BORDER_SUNKEN, // тип границы
                      const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // угол графика
                      const color          clr=clrRed,          // цвет плоской
                      const ENUM_LINE_STYLE style=STYLE_SOLID,    // стиль плоской
                      const int            line_width=1,          // толщина плоской
                      const bool            back=false,           // на заднем плане
                      const bool            selection=false,        // выделить для
                      const bool            hidden=true,           // скрыт в списке
                      const long            z_order=0)           // приоритет на
{
//--- сбросим значение ошибки
ResetLastError();
//--- создадим прямоугольную метку
if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE_LABEL,sub_window,0,0))
{
Print(__FUNCTION__,
": не удалось создать прямоугольную метку! Код ошибки = ",GetLastError());
return(false);
}
//--- установим координаты метки
ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- установим размеры метки
ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим цвет фона
ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- установим тип границы
ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border);
//--- установим угол графика, относительно которого будут определяться координаты точек
ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- установим цвет плоской рамки (в режиме Flat)
ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии плоской рамки

```

```

ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину плоской границы
ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,line_width);
//--- отобразим на переднем (false) или заднем (true) плане
ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установим приоритет на получение события нажатия мыши на графике
ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
return(true);
}

//+-----+
//| Перемещает прямоугольную метку |
//+-----+
bool RectLabelMove(const long chart_ID=0,           // ID графика
                   const string name="RectLabel", // имя метки
                   const int    x=0,             // координата по оси X
                   const int    y=0)            // координата по оси Y
{
//--- сбросим значение ошибки
ResetLastError();
//--- переместим прямоугольную метку
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
{
Print(__FUNCTION__,
      ": не удалось переместить X-координату метки! Код ошибки = ",GetLastError
      return(false);
}
if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
{
Print(__FUNCTION__,
      ": не удалось переместить Y-координату метки! Код ошибки = ",GetLastError
      return(false);
}
//--- успешное выполнение
return(true);
}

//+-----+
//| Изменяет размер прямоугольной метки |
//+-----+
bool RectLabelChangeSize(const long chart_ID=0,        // ID графика
                        const string name="RectLabel", // имя метки
                        const int   width=50,         // ширина метки
                        const int   height=18)        // высота метки
{

```

```

//--- сбросим значение ошибки
ResetLastError();

//--- изменим размеры метки
if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
{
    Print(__FUNCTION__,
          ": не удалось изменить ширину метки! Код ошибки = ", GetLastError());
    return(false);
}

if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
{
    Print(__FUNCTION__,
          ": не удалось изменить высоту метки! Код ошибки = ", GetLastError());
    return(false);
}

//--- успешное выполнение
return(true);
}

//+-----+
// | Изменяет тип границы прямоугольной метки
//+-----+
bool RectLabelChangeBorderType(const long      chart_ID=0,           // ID графика
                                const string    name="RectLabel", // имя метки
                                const ENUM_BORDER_TYPE border=BORDER_SUNKEN) // тип границы
{
    //--- сбросим значение ошибки
    ResetLastError();

    //--- изменим тип рамки
    if(!ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border))
    {
        Print(__FUNCTION__,
              ": не удалось изменить тип границы! Код ошибки = ", GetLastError());
        return(false);
    }

    //--- успешное выполнение
    return(true);
}

//+-----+
// | Удаляет прямоугольную метку
//+-----+
bool RectLabelDelete(const long   chart_ID=0,           // ID графика
                     const string name="RectLabel") // имя метки
{
    //--- сбросим значение ошибки
    ResetLastError();

    //--- удалим метку
    if(!ObjectDelete(chart_ID,name))
    {
        Print(__FUNCTION__,
              ": не удалось удалить метку! Код ошибки = ", GetLastError());
    }
}

```

```

        ": не удалось удалить прямоугольную метку! Код ошибки = ", GetLastError());
    return(false);
}
//--- успешное выполнение
return(true);
}
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- размеры окна графика
long x_distance;
long y_distance;
//--- определим размеры окна
if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
{
    Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
    return;
}
if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
{
    Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
    return;
}
//--- определим координаты прямоугольной метки
int x=(int)x_distance/4;
int y=(int)y_distance/4;
//--- установим размеры метки
int width=(int)x_distance/4;
int height=(int)y_distance/4;
//--- создадим прямоугольную метку
if(!RectLabelCreate(0,IvpName,0,x,y,width,height,IvpBackColor,IvpBorder,IvpCorner,
IvpColor,IvpStyle,IvpLineWidth,IvpBack,IvpSelection,IvpHidden,IvpZOrder))
{
    return;
}
//--- перерисуем график и подождем одну секунду
ChartRedraw();
Sleep(1000);
//--- изменим размер прямоугольной метки
int steps=(int)MathMin(x_distance/4,y_distance/4);
for(int i=0;i<steps;i++)
{
    //--- изменим размеры
    width+=1;
    height+=1;
    if(!RectLabelChangeSize(0,IvpName,width,height))
        return;
}
}

```

```
//--- проверим факт принудительного завершения скрипта
if(IsStopped())
    return;
//--- перерисуем график и подождем 0.01 секунды
ChartRedraw();
Sleep(10);
}

//--- задержка в 1 секунду
Sleep(1000);

//--- изменим тип рамки
if(!RectLabelChangeBorderType(0, InpName, BORDER_RAISED))
    return;
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- изменим тип рамки
if(!RectLabelChangeBorderType(0, InpName, BORDER_SUNKEN))
    return;
//--- перерисуем график и подождем 1 секунду
ChartRedraw();
Sleep(1000);

//--- удалим метку
RectLabelDelete(0, InpName);
ChartRedraw();

//--- подождем 1 секунду
Sleep(1000);

//---
```

## Свойства объектов

Графические объекты могут иметь множество свойств в зависимости от типа объекта. Установка и получение значений свойств объектов производится соответствующими [функциями по работе с графическими объектами](#).

Все объекты, используемые в техническом анализе, имеют привязку на графиках по координатам цены и времени - трендовая линия, каналы, инструменты Фибоначчи и т.д. Но есть ряд вспомогательных объектов, предназначенных для улучшения интерфейса, которые имеют привязку к видимой всегда части графика (основное окно графика или подокна индикаторов):

Объект	Идентификатор	X/Y	Width/Height	Date/Price	<u>OBJPROP_CORNER</u>	<u>OBJPROP_ANCHOR</u>	<u>OBJPROP_ANGLE</u>
Text	<u>OBJ_TEXT</u>	—	—	Да	—	Да	Да
Label	<u>OBJ_LABEL</u>	Да	Да (только для чтения)	—	Да	Да	Да
Button	<u>OBJ_BUTTON</u>	Да	Да	—	Да	—	—
Bitmap	<u>OBJ_BITMAP</u>	—	Да (только для чтения)	Да	—	Да	—
Bitmap Label	<u>OBJ_BITMAP_LABEL</u>	Да	Да (только для чтения)	—	Да	Да	—
Edit	<u>OBJ_EDIT</u>	Да	Да	—	Да	—	—
Rectangle Label	<u>OBJ_RECTANGLE_LABEL</u>	Да	Да	—	Да	—	—

В таблице использованы следующие обозначения:

- **X/Y** - координаты точки привязки задаются в пикселях относительно одного из углов графика;
- **Width/Height** - объекты имеет ширину и высоту. Если указано "только для чтения", то это означает, что значения ширины и высоты вычисляются только после отрисовки объекта на графике;
- **Date/Price** - координаты точки привязки задаются парой дата/цена;
- **OBJPROP\_CORNER** - задаёт угол графика, относительно которого указываются координаты точки привязки. Может быть одним из 4-х значений перечисления [ENUM\\_BASE\\_CORNER](#);
- **OBJPROP\_ANCHOR** - задаёт положение точки привязки в самом объекте, и может быть одним из 9-ти значений перечисления [ENUM\\_ANCHOR\\_POINT](#). Именно от этой точки до выбранного угла графика указываются координаты в пикселях;

- **OBJPROP\_ANGLE** - задаёт угол поворота объекта против часовой стрелки.

Функции, задающие свойства графических объектов, а также операции создания [ObjectCreate\(\)](#) и перемещения [ObjectMove\(\)](#) объектов на графике фактически служат для отправки команд графику. При успешном выполнении этих функций команда попадает в общую очередь событий графика. Визуальное изменение свойств графических объектов производится в процессе обработки очереди событий данного графика.

По этой причине не следует ожидать немедленного визуального обновления графических объектов после вызова данных функций. В общем случае обновление графических объектов на чарте производится терминалом автоматически по событиям изменения - поступление новой котировки, изменения размера окна графика и т.д. Для принудительного обновления графических объектов используйте команду на перерисовку графика [ChartRedraw\(\)](#).

Для функций [ObjectSetInteger\(\)](#) и [ObjectGetInteger\(\)](#)

#### ENUM\_OBJECT\_PROPERTY\_INTEGER

Идентификатор	Описание	Тип свойства
OBJPROP_COLOR	Цвет	color
OBJPROP_STYLE	Стиль	<a href="#">ENUM_LINE_STYLE</a>
OBJPROP_WIDTH	Толщина линии	int
OBJPROP_BACK	Объект на заднем плане	bool
OBJPROP_ZORDER	Приоритет графического объекта на получение события нажатия мышки на графике ( <a href="#">CHARTEVENT_CLICK</a> ). По умолчанию при создании значение выставляется равным нулю, но при необходимости можно повысить приоритет. При наложении объектов друг на друга событие CHARTEVENT_CLICK получит только один объект, чей приоритет выше остальных.	long
OBJPROP_FILL	Заливка объекта цветом (для OBJ_RECTANGLE, OBJ_TRIANGLE, OBJ_ELLIPSE, OBJ_CHANNEL, OBJ_STDDEVCHANNEL, OBJ_REGRESSION)	bool
OBJPROP_HIDDEN	Запрет на показ имени графического объекта в списке объектов из меню терминала "Графики" -	bool

	"Объекты" - "Список объектов". Значение true позволяет скрыть ненужный для пользователя объект из списка. По умолчанию true устанавливается для объектов, которые отображают события календаря, историю торговли, а также для <a href="#">созданных из MQL5-программы</a> . Для того чтобы увидеть такие <a href="#">графические объекты</a> и получить доступ к их свойствам, нужно нажать кнопку "Все" в окне "Список объектов".	
OBJPROP_SELECTED	Выделенность объекта	bool
OBJPROP_READONLY	Возможность редактирования текста в объекте Edit	bool
OBJPROP_TYPE	Тип объекта	<a href="#">ENUM_OBJECT</a> r/o
OBJPROP_TIME	Координата времени	datetime модификатор=номер точки привязки
OBJPROP_SELECTABLE	Доступность объекта	bool
OBJPROP_CREATETIME	Время создания объекта	datetime r/o
OBJPROP_LEVELS	Количество уровней	int
OBJPROP_LEVELCOLOR	Цвет линии-уровня	color модификатор=номер уровня
OBJPROP_LEVELSTYLE	Стиль линии-уровня	<a href="#">ENUM_LINE_STYLE</a> модификатор=номер уровня
OBJPROP_LEVELWIDTH	Толщина линии-уровня	int модификатор=номер уровня
OBJPROP_ALIGN	Горизонтальное выравнивание текста в объекте "Поле ввода" (OBJ_EDIT)	<a href="#">ENUM_ALIGN_MODE</a>
OBJPROP_FONTSIZE	Размер шрифта	int
OBJPROP_RAY_LEFT	Луч продолжается влево	bool
OBJPROP_RAY_RIGHT	Луч продолжается вправо	bool
OBJPROP_RAY	Вертикальная линия продолжается на все окна	bool

	графика	
OBJPROP_ELLIPSE	Отображение полного эллипса для объекта "Дуги Фибоначчи" ( <a href="#">OBJ_FIBOARC</a> )	bool
OBJPROP_ARROWCODE	Код стрелки для объекта "Стрелка"	char
OBJPROP_TIMEFRAMES	Видимость объекта на таймфреймах	набор флагов <a href="#">flags</a>
OBJPROP_ANCHOR	Положение точки привязки графического объекта	<a href="#">ENUM_ARROW_ANCHOR</a> (для OBJ_ARROW), <a href="#">ENUM_ANCHOR_POINT</a> (для OBJ_LABEL, OBJ_BITMAP_LABEL и OBJ_TEXT)
OBJPROP_XDISTANCE	Дистанция в пикселях по оси X от угла привязки (см. <a href="#">примечание</a> )	int
OBJPROP_YDISTANCE	Дистанция в пикселях по оси Y от угла привязки (см. <a href="#">примечание</a> )	int
OBJPROP_DIRECTION	Тренд объекта Ганна	<a href="#">ENUM_GANN_DIRECTION</a>
OBJPROP_DEGREE	Уровень волновой разметки Эллиота	<a href="#">ENUM_ELLIOT_WAVE_DEGREE</a>
OBJPROP_DRAWLINES	Отображение линий для волновой разметки Эллиота	bool
OBJPROP_STATE	Состояние кнопки (Нажата/Отжата)	bool
OBJPROP_CHART_ID	Идентификатор объекта "График" ( <a href="#">OBJ_CHART</a> ). Позволяет работать со свойствами этого объекта как с обычным графиком с помощью функций из раздела <a href="#">Операции с графиками</a> , но есть некоторые <a href="#">исключения</a> .	long r/o
OBJPROP_XSIZE	Ширина объекта по оси X в пикселях. Задается для объектов OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL.	int

OBJPROP_YSIZE	Высота объекта по оси Y в пикселях. Задается для объектов OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL.	int
OBJPROP_XOFFSET	X-координата левого верхнего угла <u>прямоугольной области видимости</u> в графических объектах "Графическая метка" и "Рисунок" (OBJ_BITMAP_LABEL и OBJ_BITMAP). Значение задается в пикселях относительного верхнего левого угла исходного изображения.	int
OBJPROP_YOFFSET	Y-координата левого верхнего угла <u>прямоугольной области видимости</u> в графических объектах "Графическая метка" и "Рисунок" (OBJ_BITMAP_LABEL и OBJ_BITMAP). Значение задается в пикселях относительного верхнего левого угла исходного изображения.	int
OBJPROP_PERIOD	Период для объекта "График"	<a href="#">ENUM_TIMEFRAMES</a>
OBJPROP_DATE_SCALE	Признак отображения шкалы времени для объекта "График"	bool
OBJPROP_PRICE_SCALE	Признак отображения ценовой шкалы для объекта "График"	bool
OBJPROP_CHART_SCALE	Масштаб для объекта "График"	int значение в диапазоне 0-5
OBJPROP_BGCOLOR	Цвет фона для OBJ_EDIT, OBJ_BUTTON, OBJ_RECTANGLE_LABEL	color
OBJPROP_CORNER	Угол графика для привязки графического объекта	<a href="#">ENUM_BASE_CORNER</a>

OBJPROP_BORDER_TYPE	Тип рамки для объекта "Прямоугольная рамка"	<a href="#">ENUM_BORDER_TYPE</a>
OBJPROP_BORDER_COLOR	Цвет рамки для объекта OBJ_EDIT и OBJ_BUTTON	color

При применении [операций с графиками](#) для объекта "График" ([OBJ\\_CHART](#)) действуют следующие ограничения:

- нельзя закрыть с помощью [ChartClose\(\)](#);
- нельзя поменять символ/период с помощью функции [ChartSetSymbolPeriod\(\)](#);
- не работают свойства CHART\_SCALE, CHART\_BRING\_TO\_TOP, CHART\_SHOW\_DATE\_SCALE и CHART\_SHOW\_PRICE\_SCALE ([ENUM\\_CHART\\_PROPERTY\\_INTEGER](#)).

Для объектов [OBJ\\_BITMAP\\_LABEL](#) и [OBJ\\_BITMAP](#) программным путем можно установить специальный режим показа изображения. В этом режиме показывается только та часть исходного изображения, на которую накладывается прямоугольная область видимости, остальная часть картинки становится невидимой. Размеры области видимости необходимо установить с помощью свойств OBJPROP\_XSIZE и OBJPROP\_YSIZE. Область видимости можно "перемещать" только в пределах исходного изображения с помощью свойств OBJPROP\_XOFFSET и OBJPROP\_YOFFSET.

Для объектов с фиксированными размерами: [OBJ\\_BUTTON](#), [OBJ\\_RECTANGLE\\_LABEL](#), [OBJ\\_EDIT](#) и [OBJ\\_CHART](#) свойства OBJPROP\_XDISTANCE и OBJPROP\_YDISTANCE задают положение левой верхней точки объекта относительно угла графика (OBJPROP\_CORNER), от которого будут отсчитываться координаты X и Y в пикселях.

Для функций [ObjectSetDouble\(\)](#) и [ObjectGetDouble\(\)](#)

#### ENUM\_OBJECT\_PROPERTY\_DOUBLE

Идентификатор	Описание	Тип свойства
OBJPROP_PRICE	Координата цены	double модификатор=номер точки привязки
OBJPROP_LEVELVALUE	Значение уровня	double модификатор=номер уровня
OBJPROP_SCALE	Масштаб (свойство объектов Ганна и объекта "Дуги Фибоначчи")	double
OBJPROP_ANGLE	Угол. Для объектов с еще не заданным углом, созданных из программы, значение равно <a href="#">EMPTY_VALUE</a>	double

OBJPROP_DEVIATION	Отклонение для канала стандартного отклонения	double
-------------------	---	--------

Для функций [ObjectSetString\(\)](#) и [ObjectGetString\(\)](#)

#### ENUM\_OBJECT\_PROPERTY\_STRING

Идентификатор	Описание	Тип свойства
OBJPROP_NAME	Имя объекта	string
OBJPROP_TEXT	Описание объекта (текст, содержащийся в объекте)	string
OBJPROP_TOOLTIP	Текст всплывающей подсказки. Если свойство не задано, то показывается подсказка, автоматически сформированная терминалом. Можно отключить показ подсказки, установив для нее значение "\n" (перевод строки)	string
OBJPROP_LEVELTEXT	Описание уровня	string модификатор=номер уровня
OBJPROP_FONT	Шрифт	string
OBJPROP_BMPFILE	Имя BMP-файла для объекта "Графическая метка". Смотри также <a href="#">Ресурсы</a>	string модификатор: 0-состояние ON, 1-состояние OFF
OBJPROP_SYMBOL	Символ для объекта "График"	string

Для объекта OBJ\_RECTANGLE\_LABEL ("Прямоугольная метка") можно задать один из трех видов отображения, которым соответствуют значения из перечисления ENUM\_BORDER\_TYPE.

#### ENUM\_BORDER\_TYPE

Идентификатор	Описание
BORDER_FLAT	Плоский вид
BORDER_RAISED	Выпуклый вид
BORDER_SUNKEN	Вогнутый вид

Для объекта OBJ\_EDIT ("Поле ввода") и для функции [ChartScreenShot\(\)](#) можно указать тип выравнивания по горизонтали с помощью значений перечисления ENUM\_ALIGN\_MODE.

#### ENUM\_ALIGN\_MODE

Идентификатор	Описание
ALIGN_LEFT	Выравнивание по левой границе
ALIGN_CENTER	Выравнивание по центру (только для объекта "Поле ввода")
ALIGN_RIGHT	Выравнивание по правой границе

**Пример:**

```
#define UP          "\x0431"
//-----
//| Script program start function
//+-----+
void OnStart()
{
//---
    string label_name="my_OBJ_LABEL_object";
    if(ObjectFind(0,label_name)<0)
    {
        Print("Object ",label_name," not found. Error code = ",GetLastError());
        //--- создадим объект Label
        ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
        //--- установим координату X
        ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
        //--- установим координату Y
        ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
        //--- зададим цвет текста
        ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrWhite);
        //--- установим текст для объекта Label
        ObjectSetString(0,label_name,OBJPROP_TEXT,UP);
        //--- установим шрифт надписи
        ObjectSetString(0,label_name,OBJPROP_FONT,"Wingdings");
        //--- установим размер шрифта
        ObjectSetInteger(0,label_name,OBJPROP_FONTSIZE,10);
        //--- повернем на 45 градусов по часовой стрелке
        ObjectSetDouble(0,label_name,OBJPROP_ANGLE,-45);
        //--- запретим выделение объекта мышкой
        ObjectSetInteger(0,label_name,OBJPROP_SELECTABLE,false);
        //--- отрисуем на графике
        ChartRedraw(0);
    }
}
```

## Способы привязки объектов

Графические объекты Text, Label, Bitmap и Bitmap Label (OBJ\_TEXT, OBJ\_LABEL, OBJ\_BITMAP и OBJ\_BITMAP\_LABEL) могут иметь один из 9 различных способов привязки своих координат, задаваемых свойством OBJPROP\_ANCHOR.

Объект	Идентификатор	X/Y	Width/Height	Date/Price	<u>OBJPR_OP_CORNER</u>	<u>OBJPR_OP_ANCHOR</u>	<u>OBJPR_OPENGL</u>
Text	<u>OBJ_TEXT</u>	—	—	Да	—	Да	Да
Label	<u>OBJ_LABEL</u>	Да	Да (только для чтения)	—	Да	Да	Да
Button	<u>OBJ_BUTTON</u>	Да	Да	—	Да	—	—
Bitmap	<u>OBJ_BITMAP</u>	—	Да (только для чтения)	Да	—	Да	—
Bitmap Label	<u>OBJ_BITMAP_LABEL</u>	Да	Да (только для чтения)	—	Да	Да	—
Edit	<u>OBJ_EDIT</u>	Да	Да	—	Да	—	—
Rectangle Label	<u>OBJ_RECTANGLE_LABEL</u>	Да	Да	—	Да	—	—

В таблице использованы следующие обозначения:

- **X/Y** - координаты точки привязки задаются в пикселях относительно одного из углов графика;
- **Width/Height** - объекты имеет ширину и высоту. Если указано "только для чтения", то это означает, что значения ширины и высоты вычисляются только после отрисовки объекта на графике;
- **Date/Price** - координаты точки привязки задаются парой дата/цена;
- **OBJPROP\_CORNER** - задаёт угол графика, относительно которого указываются координаты точки привязки. Может быть одним из 4-х значений перечисления ENUM\_BASE\_CORNER;
- **OBJPROP\_ANCHOR** - задаёт положение точки привязки в самом объекте, и может быть одним из 9-ти значений перечисления ENUM\_ANCHOR\_POINT. Именно от этой точки до выбранного угла графика указываются координаты в пикселях;
- **OBJPROP\_ANGLE** - задаёт угол поворота объекта против часовой стрелки.

Указать нужный вариант можно с помощью функции `ObjectSetInteger(handle_графика, имя_объекта, OBJPROP_ANCHOR, способ_привязки)`, где способ\_привязки - одно из значений перечисления ENUM\_ANCHOR\_POINT.

#### ENUM\_ANCHOR\_POINT

Идентификатор	Описание
ANCHOR_LEFT_UPPER	Точка привязки в левом верхнем углу
ANCHOR_LEFT	Точка привязки слева по центру
ANCHOR_LEFT_LOWER	Точка привязки в левом нижнем углу
ANCHOR_LOWER	Точка привязки снизу по центру
ANCHOR_RIGHT_LOWER	Точка привязки в правом нижнем углу
ANCHOR_RIGHT	Точка привязки справа по центру
ANCHOR_RIGHT_UPPER	Точка привязки в правом верхнем углу
ANCHOR_UPPER	Точка привязки сверху по центру
ANCHOR_CENTER	Точка привязки строго по центру объекта

Объекты `OBJ_BUTTON`, `OBJ_RECTANGLE_LABEL`, `OBJ_EDIT` и `OBJ_CHART` имеют фиксированную точку привязки в левом верхнем углу (ANCHOR\_LEFT\_UPPER).

#### Пример:

```

string text_name="my_OBJ_TEXT_object";
if(ObjectFind(0,text_name)<0)
{
    Print("Object ",text_name," not found. Error code = ",GetLastError());
    //--- получим максимальную цену графика
    double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
    //--- создадим объект Label
    ObjectCreate(0,text_name,OBJ_TEXT,0,TimeCurrent(),chart_max_price);
    //--- зададим цвет текста
    ObjectSetInteger(0,text_name,OBJPROP_COLOR,clrWhite);
    //--- зададим цвет фона
    ObjectSetInteger(0,text_name,OBJPROP_BGCOLOR,clrGreen);
    //--- установим текст для объекта Label
    ObjectSetString(0,text_name,OBJPROP_TEXT,TimeToString(TimeCurrent()));
    //--- установим шрифт надписи
    ObjectSetString(0,text_name,OBJPROP_FONT,"Trebuchet MS");
    //--- установим размер шрифта
    ObjectSetInteger(0,text_name,OBJPROP_FONTSIZE,10);
    //--- установим привязку к правому верхнему углу
    ObjectSetInteger(0,text_name,OBJPROP_ANCHOR,ANCHOR_RIGHT_UPPER);
    //--- повернем на 90 градусов против часовой стрелке
    ObjectSetDouble(0,text_name,OBJPROP_ANGLE,90);
    //--- запретим выделение объекта мышкой
}

```

```

ObjectSetInteger(0, text_name, OBJPROP_SELECTABLE, false);
//--- отрисуем на графике
ChartRedraw(0);
}

```

Графические объекты Arrow (OBJ\_ARROW) имеют только 2 способа привязки своих координат. Идентификаторы перечислены в ENUM\_ARROW\_ANCHOR.

#### ENUM\_ARROW\_ANCHOR

Идентификатор	Описание
ANCHOR_TOP	Точка привязки для стрелки находится сверху
ANCHOR_BOTTOM	Точка привязки для стрелки находится внизу

#### Пример:

```

void OnStart()
{
//--- служебные массивы
double Ups[], Downs[];
datetime Time[];

//--- установим для массивов признак таймсерии
ArraySetAsSeries(Ups, true);
ArraySetAsSeries(Downs, true);
ArraySetAsSeries(Time, true);

//--- создадим хэндл индикатора Fractals
int FractalsHandle=iFractals(NULL, 0);
Print("FractalsHandle = ", FractalsHandle);

//--- сбросим код ошибки
ResetLastError();

//--- попытаемся скопировать значения индикатора
int copied=CopyBuffer(FractalsHandle, 0, 0, 1000, Ups);
if(copied<=0)
{
Print("Не удалось скопировать верхние фракталы. Error = ", GetLastError());
return;
}

ResetLastError();

//--- попытаемся скопировать значения индикатора
copied=CopyBuffer(FractalsHandle, 1, 0, 1000, Downs);
if(copied<=0)
{
Print("Не удалось скопировать нижние фракталы. Error = ", GetLastError());
return;
}

ResetLastError();

//--- скопируем таймсерию, содержащую время открытия последних 1000 баров
copied=CopyTime(NULL, 0, 0, 1000, Time);
}

```

```

if(copied<=0)
{
    Print("Не удалось скопировать времена открытия за последние 1000 баров");
    return;
}

int upcounter=0,downcounter=0; // будем в них подсчитывать количество стрелок
bool created;// будем получать результат попытки создания объекта
for(int i=2;i<copied;i++)// пробежимся по значениям индикатора iFractals
{
    if(Ups[i]!=EMPTY_VALUE)// нашли верхний фрактал
    {
        if(upcounter<10)// создаем не более 10 объектов "вверх"
        {
            //--- попробуем создать объект "вверх"
            created=ObjectCreate(0,string(Time[i]),OBJ_ARROW_THUMB_UP,0,Time[i],Ups[i]);
            if(created)// если создался - сделаем ему тюнинг
            {
                //--- точка привязки снизу, чтобы не наезжать на бар
                ObjectSetInteger(0,string(Time[i]),OBJPROP_ANCHOR,ANCHOR_BOTTOM);
                //--- последний штрих - покрасим
                ObjectSetInteger(0,string(Time[i]),OBJPROP_COLOR,clrBlue);
                upcounter++;
            }
        }
    }
    if(Downs[i]!=EMPTY_VALUE)// нашли нижний фрактал
    {
        if(downcounter<10)// создаем не более 10 объектов "вниз"
        {
            //--- попробуем создать объект "вниз"
            created=ObjectCreate(0,string(Time[i]),OBJ_ARROW_THUMB_DOWN,0,Time[i],Downs[i]);
            if(created)// если создался - сделаем ему тюнинг
            {
                //--- точка привязки сверху, чтобы не наезжать на бар
                ObjectSetInteger(0,string(Time[i]),OBJPROP_ANCHOR,ANCHOR_TOP);
                //--- последний штрих - покрасим
                ObjectSetInteger(0,string(Time[i]),OBJPROP_COLOR,clrRed);
                downcounter++;
            }
        }
    }
}
}

```

После выполнения скрипта график будет выглядеть примерно как на этом рисунке.



## Угол графика, к которому привязан объект

Существует ряд [графических объектов](#), для которых можно задавать угол графика, относительно которого указываются координаты в пикселях. Это следующие типы объектов (в скобках указаны идентификаторы типа объекта):

- Label (OBJ\_LABEL); Текстовая метка
- Button (OBJ\_BUTTON); Кнопка
- Bitmap Label (OBJ\_BITMAP\_LABEL); Графическая метка
- Edit (OBJ\_EDIT); Поле ввода
- Rectangle Label (OBJ\_RECTANGLE\_LABEL). Прямоугольная метка

Объект	Идентификатор	X/Y	Width/Height	Date/Price	OBJPROP_CORN	OBJPR	OBJPR
					OP_CO	OP_AN	OP_AN
					RNER	CHOR	GLE
Text	<a href="#">OBJ_TEXT</a>	—	—	Да	—	Да	Да
Label	<a href="#">OBJ_LABE</a> <a href="#">L</a>	Да	Да (только для чтения)	—	Да	Да	Да
Button	<a href="#">OBJ_BUTT</a> <a href="#">ON</a>	Да	Да	—	Да	—	—
Bitmap	<a href="#">OBJ_BITMAP</a>	—	Да (только для чтения)	Да	—	Да	—
Bitmap Label	<a href="#">OBJ_BITMAP_LABEL</a>	Да	Да (только для чтения)	—	Да	Да	—
Edit	<a href="#">OBJ_EDIT</a>	Да	Да	—	Да	—	—
Rectangle Label	<a href="#">OBJ_RECTANGLE_LABEL</a>	Да	Да	—	Да	—	—

В таблице использованы следующие обозначения:

- X/Y - координаты точки привязки задаются в пикселях относительно одного из углов графика;
- Width/Height - объекты имеет ширину и высоту. Если указано "только для чтения", то это означает, что значения ширины и высоты вычисляются только после отрисовки объекта на графике;
- Date/Price - координаты точки привязки задаются парой дата/цена;
- OBJPROP\_CORNER - задаёт угол графика, относительно которого указываются координаты точки привязки. Может быть одним из 4-х значений перечисления [ENUM\\_BASE\\_CORNER](#);

- **OBJPROP\_ANCHOR** - задаёт положение точки привязки в самом объекте, и может быть одним из 9-ти значений перечисления [ENUM\\_ANCHOR\\_POINT](#). Именно от этой точки до выбранного угла графика указываются координаты в пикселях;
- **OBJPROP\_ANGLE** - задаёт угол поворота объекта против часовой стрелки.

Для того чтобы указать угол графика, от которого будут отсчитываться координаты X и Y в пикселях, необходимо воспользоваться функцией [ObjectSetInteger](#)(chartID, name, **OBJPROP\_CORNER**, chart\_corner), где:

- chartID - идентификатор графика;
- name - имя графического объекта;
- **OBJPROP\_CORNER** - идентификатор свойства для задания угла привязки;
- **chart\_corner** - требуемый угол графика, может принимать одно из значений перечисления [ENUM\\_BASE\\_CORNER](#).

#### **ENUM\_BASE\_CORNER**

Идентификатор	Описание
CORNER_LEFT_UPPER	Центр координат в левом верхнем углу графика
CORNER_LEFT_LOWER	Центр координат в левом нижнем углу графика
CORNER_RIGHT_LOWER	Центр координат в правом нижнем углу графика
CORNER_RIGHT_UPPER	Центр координат в правом верхнем углу графика

**Пример:**

```
void CreateLabel(long    chart_id,
                 string   name,
                 int      chart_corner,
                 int      anchor_point,
                 string   text_label,
                 int      x_ord,
                 int      y_ord)
{
//---
    if(ObjectCreate(chart_id, name, OBJ_LABEL, 0, 0, 0))
    {
        ObjectSetInteger(chart_id, name, OBJPROP_CORNER, chart_corner);
        ObjectSetInteger(chart_id, name, OBJPROP_ANCHOR, anchor_point);
        ObjectSetInteger(chart_id, name, OBJPROP_XDISTANCE, x_ord);
        ObjectSetInteger(chart_id, name, OBJPROP_YDISTANCE, y_ord);
        ObjectSetString(chart_id, name, OBJPROP_TEXT, text_label);
    }
    else
        Print("Не удалось создать объект OBJ_LABEL ", name, ", Код ошибки = ", GetLastError());
}
```

```
    }

//+-----+
//| Script program start function           |
//+-----+

void OnStart()
{
//---

    int height=(int)ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);
    int width=(int)ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0);
    string arrows[4]={"LEFT_UPPER","RIGHT_UPPER","RIGHT_LOWER","LEFT_LOWER"};
    CreateLabel(0,arrows[0],CORNER_LEFT_UPPER,ANCHOR_LEFT_UPPER,arrows[0],50,50);
    CreateLabel(0,arrows[1],CORNER_RIGHT_UPPER,ANCHOR_RIGHT_UPPER,arrows[1],50,50);
    CreateLabel(0,arrows[2],CORNER_RIGHT_LOWER,ANCHOR_RIGHT_LOWER,arrows[2],50,50);
    CreateLabel(0,arrows[3],CORNER_LEFT_LOWER,ANCHOR_LEFT_LOWER,arrows[3],50,50);
}
}
```

## Видимость объектов

Комбинация флагов видимости объекта определяет таймфреймы графика, на которых объект отображаем. Для установки/получения значения свойства OBJPROP\_TIMEFRAMES можно использовать функции [ObjectSetInteger\(\)](#)/[ObjectGetInteger\(\)](#).

Константа	Значение	Описание
OBJ_NO_PERIODS	0	Объект не показывается ни на одном таймфрейме
OBJ_PERIOD_M1	0x00000001	Объект рисуется на 1-минутных графиках
OBJ_PERIOD_M2	0x00000002	Объект рисуется на 2-минутных графиках
OBJ_PERIOD_M3	0x00000004	Объект рисуется на 3-минутных графиках
OBJ_PERIOD_M4	0x00000008	Объект рисуется на 4-минутных графиках
OBJ_PERIOD_M5	0x00000010	Объект рисуется на 5-минутных графиках
OBJ_PERIOD_M6	0x00000020	Объект рисуется на 6-минутных графиках
OBJ_PERIOD_M10	0x00000040	Объект рисуется на 10-минутных графиках
OBJ_PERIOD_M12	0x00000080	Объект рисуется на 12-минутных графиках
OBJ_PERIOD_M15	0x00000100	Объект рисуется на 15-минутных графиках
OBJ_PERIOD_M20	0x00000200	Объект рисуется на 20-минутных графиках
OBJ_PERIOD_M30	0x00000400	Объект рисуется на 30-минутных графиках
OBJ_PERIOD_H1	0x00000800	Объект рисуется на 1-часовых графиках
OBJ_PERIOD_H2	0x00001000	Объект рисуется на 2-часовых графиках
OBJ_PERIOD_H3	0x00002000	Объект рисуется на 3-часовых графиках
OBJ_PERIOD_H4	0x00004000	Объект рисуется на 4-часовых графиках

OBJ_PERIOD_H6	0x00008000	Объект рисуется на 6-часовых графиках
OBJ_PERIOD_H8	0x00010000	Объект рисуется на 8-часовых графиках
OBJ_PERIOD_H12	0x00020000	Объект рисуется на 12-часовых графиках
OBJ_PERIOD_D1	0x00040000	Объект рисуется на дневных графиках
OBJ_PERIOD_W1	0x00080000	Объект рисуется на недельных графиках
OBJ_PERIOD_MN1	0x00100000	Объект рисуется на месячных графиках
OBJ_ALL_PERIODS	0x001fffff	Объект рисуется на всех таймфреймах

Флаги видимости можно комбинировать с помощью символа "|", например, комбинация флагов OBJ\_PERIOD\_M10|OBJ\_PERIOD\_H4 означает, что объект будет видим на 10-минутном и 4-часовом таймфреймах.

#### Пример:

```
void OnStart()
{
//---
    string highlevel="PreviousDayHigh";
    string lowlevel="PreviousDayLow";
    double prevHigh;           // High предыдущего дня
    double prevLow;            // Low предыдущего дня
    double highs[], lows[];   // массивы для получения High и Low

//--- сбросим значение последней ошибки
    ResetLastError();
//--- получим 2 последних значения High на дневном таймфрейме
    int highsgot=CopyHigh(Symbol(), PERIOD_D1, 0, 2, highs);
    if(highsgot>0) // если копирование прошло успешно
    {
        Print("Цены High за последние 2 дня получены успешно");
        prevHigh=highs[0]; // High предыдущего дня
        Print("prevHigh = ", prevHigh);
        if(ObjectFind(0,highlevel)<0) // объект с именем highlevel не найден
            ObjectCreate(0,highlevel,OBJ_HLINE,0,0,0); // создадим объект гор. линия
//--- зададим ценовой уровень для линии highlevel
        ObjectSetDouble(0,highlevel,OBJPROP_PRICE,0,prevHigh);
//--- установим видимость только для PERIOD_M10 и PERIOD_H4
        ObjectSetInteger(0,highlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M10|OBJ_PERIOD_H4);
    }
    else
}
```

```
Print("Не удалось получить цены High за последние 2 дня, Error = ",GetLastError)

//--- сбросим значение последней ошибки
ResetLastError();

//--- получим 2 последних значения Low на дневном таймфрейме
int lowsgot=CopyLow(Symbol(),PERIOD_D1,0,2, lows);
if(lows>0) // если копирование прошло успешно
{
    Print("Цены Low за последние 2 дня получены успешно");
    prevLow=lows[0]; // Low предыдущего дня
    Print("prevLow = ",prevLow);
    if(ObjectFind(0,lowlevel)<0) // объект с именем lowlevel не найден
        ObjectCreate(0,lowlevel,OBJ_HLINE,0,0,0); // создадим объект гор. линия
    //--- зададим ценовой уровень для линии lowlevel
    ObjectSetDouble(0,lowlevel,OBJPROP_PRICE,0,prevLow);
    //--- установим видимость только для PERIOD_M10 и PERIOD_H4
    ObjectSetInteger(0,lowlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M10|OBJ_PERIOD_H4);
}
else
    Print("Не удалось получить цены Low за последние 2 дня, Error = ",GetLastError())

ChartRedraw(0); // перерисуем график принудительно
}
```

#### Смотри также

[PeriodSeconds](#), [Period](#), [Периоды графиков](#), [Дата и время](#)

## Уровни волн Эллиотта

Волны Эллиотта представлены двумя графическими объектами типов OBJ\_ELLIOTWAVE5 и OBJ\_ELLIOTWAVE3. Для задания размера волны (способа маркировки волн) используется свойство OBJPROP\_DEGREE, которому можно установить одно из значений перечисления ENUM\_ELLIOT\_WAVE\_DEGREE.

### ENUM\_ELLIOT\_WAVE\_DEGREE

Константа	Описание
ELLIOTT_GRAND_SUPERCYCLE	Главный Суперцикл (Grand Supercycle)
ELLIOTT_SUPERCYCLE	Суперцикл (Supercycle)
ELLIOTT_CYCLE	Цикл (Cycle)
ELLIOTT_PRIMARY	Первичный цикл (Primary)
ELLIOTT_INTERMEDIATE	Промежуточное звено (Intermediate)
ELLIOTT_MINOR	Второстепенный цикл (Minor)
ELLIOTT_MINUTE	Минута (Minute)
ELLIOTT_MINUETTE	Секунда (Minuette)
ELLIOTT_SUBMINUETTE	Субсекунда (Subminuette)

Пример:

```
for(int i=0;i<ObjectsTotal(0);i++)
{
    string currobj=ObjectName(0,i);
    if((ObjectGetInteger(0,currobj,OBJPROP_TYPE)==OBJ_ELLIOTWAVE3) ||
       ((ObjectGetInteger(0,currobj,OBJPROP_TYPE)==OBJ_ELLIOTWAVE5)))
    {
        //--- установим уровень разметки в INTERMEDIATE
        ObjectSetInteger(0,currobj,OBJPROP_DEGREE,ELLIOTT_INTERMEDIATE);
        //--- включим показ линий между вершинами волн
        ObjectSetInteger(0,currobj,OBJPROP_DRAWLINES,true);
        //--- установим цвет линий
        ObjectSetInteger(0,currobj,OBJPROP_COLOR,clrBlue);
        //--- установим толщину линий
        ObjectSetInteger(0,currobj,OBJPROP_WIDTH,5);
        //--- зададим описание
        ObjectSetString(0,currobj,OBJPROP_TEXT,"test script");
    }
}
```

## Объекты Ганна

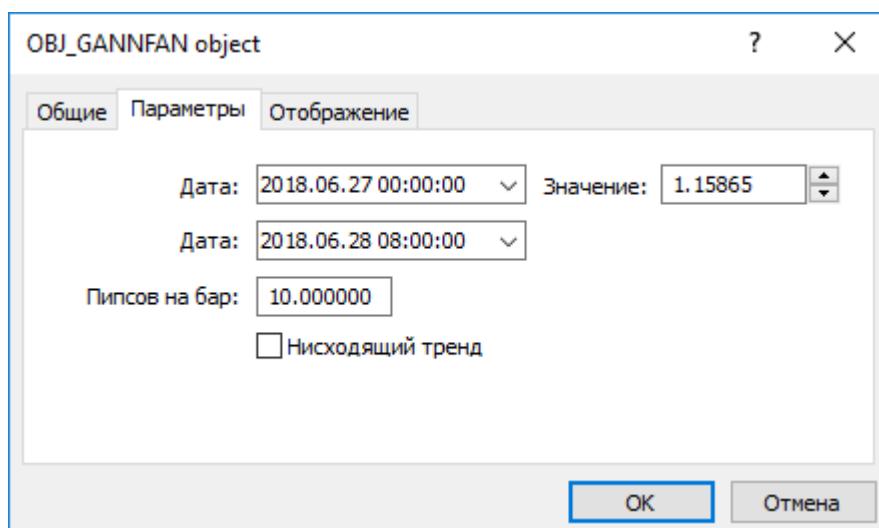
Для объектов веер Ганна (OBJ\_GANNFAN) и сетка Ганна (OBJ\_GANNGRID) можно указать одно из двух значений перечисления ENUM\_GANN\_DIRECTION, которое задает направление тренда.

### ENUM\_GANN\_DIRECTION

Константа	Описание
GANN_UP_TREND	Линия соответствует восходящему тренду
GANN_DOWN_TREND	Линия соответствует нисходящему тренду

Для установки масштаба основной линии 1x1 используется функция [ObjectSetDouble\(chart\\_handle, gann\\_object\\_name, OBJPROP\\_SCALE, scale\)](#), где:

- chart\_handle - окно графика, в котором находится объект;
- gann\_object\_name - имя объекта;
- OBJPROP\_SCALE - идентификатор свойства "Scale";
- scale - требуемый масштаб в единицах Pips/Bar.



Пример создания веера Ганна:

```
void OnStart()
{
//---
string my_gann="OBJ_GANNFAN object";
if(ObjectFind(0,my_gann)<0)// объект не найден
{
//--- сообщим о неудаче
Print("Object ",my_gann," not found. Error code = ",GetLastError());
//--- получим максимальную цену графика
double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
//--- получим минимальную цену графика
double chart_min_price=ChartGetDouble(0,CHART_PRICE_MIN,0);
//--- сколько баров на графике показано?
```

```

int bars_on_chart=ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- создадим массив, куда запишем время открытия каждого бара
datetime Time[];
//--- организуем доступ к массиву как в таймсерии
ArraySetAsSeries(Time,true);
//--- теперь скопируем в него данные для видимых на графике баров
int times=CopyTime(NULL,0,0,bars_on_chart,Time);
if(times<=0)
{
    Print("Не удалось скопировать массив со временем открытия!");
    return;
}
//--- предварительные приготовления закончены

//--- индекс центрального бара на графике
int center_bar=bars_on_chart/2;
//--- экватор графика - между максимумом и минимумом
double mean=(chart_max_price+chart_min_price)/2.0;
//--- установим координаты первой точки привязки в центр
ObjectCreate(0,my_gann,OBJ_GANNFAN,0,Time[center_bar],mean,
            //--- вторая точки привязки справа
            Time[center_bar/2],(mean+chart_min_price)/2.0);
Print("Time[center_bar] = "+(string)Time[center_bar]+"
//Print("Time[center_bar]/"+Time[center_bar]+"
//--- установим масштаб в единицах Pips/Bar
ObjectSetDouble(0,my_gann,OBJPROP_SCALE,10);
//--- установим направление тренда
ObjectSetInteger(0,my_gann,OBJPROP_DIRECTION,GANN_UP_TREND);
//--- установим толщину линий
ObjectSetInteger(0,my_gann,OBJPROP_WIDTH,1);
//--- зададим стиль линий
ObjectSetInteger(0,my_gann,OBJPROP_STYLE,STYLE_DASHDOT);
//--- и цвет линий
ObjectSetInteger(0,my_gann,OBJPROP_COLOR,clrYellowGreen);
//--- разрешим пользователю выделять объект
ObjectSetInteger(0,my_gann,OBJPROP_SELECTABLE,true);
//--- выделим его сами
ObjectSetInteger(0,my_gann,OBJPROP_SELECTED,true);
//--- отрисуем на графике
ChartRedraw(0);
}
}

```

## Набор Web-цветов

Для типа `color` определены следующие цветовые константы:

clrLightSeaGreen								clrDarkTurquoise
clrGoldrod	clrMediumSpringGreen	clrLawnGreen						
	clrOrange	clrGold	clrYellow	clrChartreuse	clrLime	clrSpringGreen	clrAqua	
clrDeepSkyBlue		clrMagenta						
		clrMediumTurquoise		clrTurquoise				clrDarkKhaki
		clrGreenYellow	clrMediumAquamarine					clrOrchid
clrMediumPurple				clrDarkGrey	clrSandyBrown			clrTan
	clrBurlyWood	clrHotPink		clrViolet		clrSkyBlue	clrLightSalmon	
clrPlum	clrKhaki	clrLightGreen	clrAquamarine	clrSilver	clrLightSkyBlue	clrLightSteelBlue	clrLightBlue	
clrPaleGreen	clrThistle	clrPowderBlue	clrPaleGoldenrod	clrPaleTurquoise	clrLightGrey	clrWheat	clrNavajoWhite	
clrMoccasin	clrLightPink	clrGainsboro	clrPeachPuff	clrPink	clrBisque	clrLightGoldenrod	clrBlanchdAlmond	
clrLemonChiffon	clrBeige	clrAntiqueWhite	clrPapayaWhip	clrCornsilk	clrLightYellow	clrLightCyan	clrLinen	
clrLavender	clrMistyRose	clrOldLace	clrWhiteSmoke	clrSeashell	clrIvory	clrHoneydew	clrAliceBlue	
clrLavenderBlush	clrMintCream	clrSnow	clrWhite					

Задавать цвет можно для объектов с помощью функции [ObjectSetInteger\(\)](#) и для пользовательских индикаторов с помощью функции [PlotIndexSetInteger\(\)](#). Для получения значения цвета служат аналогичные функции [ObjectGetInteger\(\)](#) и [PlotIndexGetInteger\(\)](#).

Пример:

```
----- indicator settings
#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots 3
#property indicator_type1 DRAW_LINE
#property indicator_type2 DRAW_LINE
#property indicator_type3 DRAW_LINE
#property indicator_color1 clrBlue
#property indicator_color2 clrRed
#property indicator_color3 clrLime
```

## Wingdings

Символы шрифта Wingdings, используемые с объектом [OBJ\\_ARROW](#):

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Установка нужного символа производится с помощью функции [ObjectSetInteger\(\)](#).

Пример:

```
void OnStart()
{
//---

    string up_arrow="up_arrow";
    datetime time=TimeCurrent();
    double lastClose[1];
    int close=CopyClose(Symbol(),Period(),0,1,lastClose);           // получим цену Close
//--- если цена получена
    if(close>0)
    {
        ObjectCreate(0,up_arrow,OBJ_ARROW,0,0,0,0);                // создадим стрелку
        ObjectSetInteger(0,up_arrow,OBJPROP_ARROWCODE,241);         // установим код стрелки
        ObjectSetInteger(0,up_arrow,OBJPROP_TIME,time);              // зададим время
        ObjectSetDouble(0,up_arrow,OBJPROP_PRICE,lastClose[0]);     // зададим цену
        ChartRedraw(0);                                            // перерисуем окно
    }
    else
        Print("Не удалось получить последнюю цену Close!");
}
```

## Константы индикаторов

Предопределены 37 стандартных [технических индикаторов](#), которые можно использовать в программах на языке MQL5. Кроме того, есть возможность создавать свои собственные пользовательские индикаторы с помощью функции [iCustom\(\)](#). Все необходимые для этого константы разделены на 5 групп:

- [Ценовые константы](#) - для выбора типа цены или объема, по которым считается индикатор;
- [Методы скользящих](#) - встроенные методы сглаживания, используемые в индикаторах;
- [Линии индикаторов](#) - идентификаторы индикаторных буферов при доступе к значениям индикаторов с помощью функции [CopyBuffer\(\)](#);
- [Стили рисования](#) - для указания одного из 18 видов отрисовки и задания стиля рисования линии;
- [Свойства пользовательских индикаторов](#) - используются в функциях для работы с [пользовательскими индикаторами](#);
- [Типы индикаторов](#) - служат для указания типа технического индикатора при создании хэндла с помощью функции [IndicatorCreate\(\)](#);
- [Идентификаторы типов данных](#) - используются для задания типа данных, передаваемых массивом типа [MqlParam](#) в функции [IndicatorCreate\(\)](#).

## Ценовые константы

Технические индикаторы требуют для своих расчетов указания значений цен и/или значений объемов, на которых они будут считаться. Существуют 7 предопределенных идентификаторов перечисления ENUM\_APPLIED\_PRICE, для указания нужной ценовой базы расчетов.

### ENUM\_APPLIED\_PRICE

Идентификатор	Описание
PRICE_CLOSE	Цена закрытия
PRICE_OPEN	Цена открытия
PRICE_HIGH	Максимальная за период цена
PRICE_LOW	Минимальная за период цена
PRICE_MEDIAN	Медианная цена, (high+low)/2
PRICE_TYPICAL	Типичная цена, (high+low+close)/3
PRICE_WEIGHTED	Средневзвешенная цена, (high+low+close+close)/4

Если в расчетах используется объем, то нужно указать одно из двух значений перечисления ENUM\_APPLIED\_VOLUME.

### ENUM\_APPLIED\_VOLUME

Идентификатор	Описание
VOLUME_TICK	Тиковый объем
VOLUME_REAL	Торговый объем

Технический индикатор [iStochastic\(\)](#) имеет два варианта расчета, которые могут использовать:

- либо только цены Close;
- либо цены High и Low.

Для выбора нужного варианта расчета нужно указать одно из значений перечисления ENUM\_STO\_PRICE.

### ENUM\_STO\_PRICE

Идентификатор	Описание
STO_LOWHIGH	Построение по ценам Low/High
STO_CLOSECLOSE	Построение по ценам Close/Close

Если технический индикатор для своих расчетов использует ценовые данные, тип которых задается перечислением ENUM\_APPLIED\_PRICE, то в качестве входного ценового ряда можно указывать хэндл любого индикатора (встроенного в терминал или написанного пользователем). В этом случае для расчетов будут использованы значения нулевого буфера индикатора. Это

позволяет легко строить значения одного индикатора по значениям другого индикатора. Хэндл пользовательского индикатора создается вызовом функции [iCustom\(\)](#).

**Пример:**

```
#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 2
//--- input parameters
input int      RSIperiod=14;           // период для вычисления RSI
input int      Smooth=8;                // период сглаживания RSI
input ENUM_MA_METHOD meth=MODE_SMMA; // метод сглаживания
//--- plot RSI
#property indicator_label1 "RSI"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- plot RSI_Smoothed
#property indicator_label2 "RSI_Smoothed"
#property indicator_type2 DRAW_LINE
#property indicator_color2 clrNavy
#property indicator_style2 STYLE_SOLID
#property indicator_width2 1
//--- indicator buffers
double        RSIBuffer[];           // здесь мы будем хранить значения RSI
double        RSI_SmoothedBuffer[]; // здесь будут сглаженные значения RSI
int          RSIhandle;              // дескриптор на индикатор RSI
//+-----+
//| Custom indicator initialization function
//+-----+
void OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,RSIBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,RSI_SmoothedBuffer,INDICATOR_DATA);
    IndicatorSetString(INDICATOR_SHORTNAME,"iRSI");
    IndicatorSetInteger(INDICATOR_DIGITS,2);
//---
    RSIhandle=iRSI(NULL,0,RSIperiod,PRICE_CLOSE);
//---
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const int begin,
               const double &price[])

```

```
)  
  
{  
//--- сбросим в ноль значение последней ошибки  
ResetLastError();  
//--- получим данные индикатора RSI в массив RSIBuffer[]  
int copied=CopyBuffer(RSIhandle,0,0,rates_total,RSIBuffer);  
if(copied<=0)  
{  
Print("Не удалось скопировать значения индикатора RSI. Error = ",  
      GetLastError(),","copied = ",copied);  
return(0);  
}  
//--- создадим индикатор средней по значениям индикатора RSI  
int RSI_MA_handle=iMA(NULL,0,Smooth,0,meth,RSIhandle);  
copied=CopyBuffer(RSI_MA_handle,0,0,rates_total,RSI_SmoothedBuffer);  
if(copied<=0)  
{  
Print("Не удалось скопировать сглаженный индикатор RSI. Error = ",  
      GetLastError(),","copied = ",copied);  
return(0);  
}  
//--- return value of prev_calculated for next call  
return(rates_total);  
}
```

## Методы скользящих

В основе многих технических индикаторов лежат те или иные методы сглаживания ценовых серий. Некоторые стандартные технические индикаторы требуют указания типа сглаживания в качестве параметра. Для указания нужного типа сглаживания служат идентификаторы, перечисленные в перечислении ENUM\_MA\_METHOD.

### ENUM\_MA\_METHOD

Идентификатор	Описание
MODE_SMA	Простое усреднение
MODE_EMA	Экспоненциальное усреднение
MODE_SMMA	Сглаженное усреднение
MODE_LWMA	Линейно-взвешенное усреднение

Пример:

```
double ExtJaws[];
double ExtTeeth[];
double ExtLips[];
//---- handles for moving averages
int ExtJawsHandle;
int ExtTeethHandle;
int ExtLipsHandle;
//--- get MA's handles
ExtJawsHandle=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN);
ExtTeethHandle=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN);
ExtLipsHandle=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN);
```

## Линии индикаторов

Некоторые [технические индикаторы](#) имеют несколько отрисовываемых на графике буферов. Нумерация индикаторных буферов начинается с 0. При копировании значений индикатора функцией [CopyBuffer\(\)](#) в массив типа double для некоторых индикаторов можно указывать не номер копируемого буфера, а идентификатор этого буфера.

Идентификаторы линий индикаторов, допустимых при копировании значений индикаторов [iMACD\(\)](#), [iRVI\(\)](#) и [iStochastic\(\)](#)

Константа	Значение	Описание
MAIN_LINE	0	Основная линия
SIGNAL_LINE	1	Сигнальная линия

Идентификаторы линий индикаторов, допустимых при копировании значений индикаторов [ADX\(\)](#) и [ADWX\(\)](#)

Константа	Значение	Описание
MAIN_LINE	0	Основная линия
PLUSDI_LINE	1	Линия +DI
MINUSDI_LINE	2	Линия -DI

Идентификаторы линий индикаторов допустимых при копировании значений индикатора [iBands\(\)](#)

Константа	Значение	Описание
BASE_LINE	0	Основная линия
UPPER_BAND	1	Верхняя граница
LOWER_BAND	2	Нижняя граница

Идентификаторы линий индикаторов, допустимых при копировании значений индикаторов [iEnvelopes\(\)](#) и [iFractals\(\)](#)

Константа	Значение	Описание
UPPER_LINE	0	Верхняя линия
LOWER_LINE	1	Нижняя линия

Идентификаторы линий индикаторов, допустимых при копировании значений индикатора [iGator\(\)](#)

Константа	Значение	Описание
UPPER_HISTOGRAM	0	Верхняя гистограмма
LOWER_HISTOGRAM	2	Нижняя гистограмма

Идентификаторы линий индикаторов, допустимых при копировании значений индикатора [iAlligator\(\)](#)

Константа	Значение	Описание
GATORJAW_LINE	0	Линия челюстей
GATORTEETH_LINE	1	Линия зубов
GATORLIPS_LINE	2	Линия губ

Идентификаторы линий индикаторов, допустимых при копировании значений индикатора [ilchimoku\(\)](#)

Константа	Значение	Описание
TENKANSEN_LINE	0	Линия Tenkan-sen
KIJUNSEN_LINE	1	Линия Kijun-sen
SENKOUSPANA_LINE	2	Линия Senkou Span A
SENKOUSPANB_LINE	3	Линия Senkou Span B
CHIKOUPAN_LINE	4	Линия Chikou Span

## Стили рисования

При создании [пользовательского индикатора](#) можно указать один из 18 типов графического построения (способа отображения на главном окне графика или в подокне графика), значения которых указаны в перечислении `ENUM_DRAW_TYPE`.

В одном пользовательском индикаторе допустимо использовать любые [виды построения/отрисовки индикаторов](#). Каждый вид построения требует указания от одного до пяти [глобальных массивов](#) для хранения данных, необходимых для рисования. Эти массивы данных необходимо связать с индикаторными буферами посредством функции `SetIndexBuffer()`, и указать для каждого буфера тип данных из перечисления `ENUM_INDEXBUFFER_TYPE`.

В зависимости от стиля рисования, может потребоваться от одного до четырех буферов значений (отмеченных как `INDICATOR_DATA`), а если стиль допускает динамическое чередование цвета (все стили со словом `COLOR` в наименовании), то необходим один буфер цвета (указан тип `INDICATOR_COLOR_INDEX`). Цветовой буфер всегда связывается сразу после соответствующих стилю буферов значений.

### `ENUM_DRAW_TYPE`

Идентификатор	Описание	Буферов значений	Буферов цвета
<code>DRAW_NONE</code>	Не отрисовывается	1	0
<code>DRAW_LINE</code>	Линия	1	0
<code>DRAW_SECTION</code>	Отрезки	1	0
<code>DRAW_HISTOGRAM</code>	Гистограмма от нулевой линии	1	0
<code>DRAW_HISTOGRAM2</code>	Гистограмма на двух индикаторных буферах	2	0
<code>DRAW_ARROW</code>	Отрисовка стрелками	1	0
<code>DRAW_ZIGZAG</code>	Стиль Zigzag допускает вертикальные отрезки на баре	2	0
<code>DRAW_FILLING</code>	Цветовая заливка между двумя уровнями	2	0
<code>DRAW_BARS</code>	Отображение в виде баров	4	0
<code>DRAW_CANDLES</code>	Отображение в виде свечей	4	0
<code>DRAW_COLOR_LINE</code>	Разноцветная линия	1	1
<code>DRAW_COLOR_SECTION</code>	Разноцветные отрезки	1	1

<a href="#">DRAW_COLOR_HISTOGRAM</a>	Разноцветная гистограмма от нулевой линии	1	1
<a href="#">DRAW_COLOR_HISTOGRAM2</a>	Разноцветная гистограмма на двух индикаторных буферах	2	1
<a href="#">DRAW_COLOR_ARROW</a>	Отрисовка разноцветными стрелками	1	1
<a href="#">DRAW_COLOR_ZIGZAG</a>	Разноцветный ZigZag	2	1
<a href="#">DRAW_COLOR_BARS</a>	Разноцветные бары	4	1
<a href="#">DRAW_COLOR_CANDLES</a>	Разноцветные свечи	4	1

Для уточнения отображения выбранного вида отрисовки используются идентификаторы, перечисленные в перечислениях ENUM\_PLOT\_PROPERTY.

Для функций [PlotIndexSetInteger\(\)](#) и [PlotIndexGetInteger\(\)](#)

#### ENUM\_PLOT\_PROPERTY\_INTEGER

Идентификатор	Описание	Тип свойства
PLOT_ARROW	Код стрелки для стиля DRAW_ARROW	uchar
PLOT_ARROW_SHIFT	Смещение стрелок по вертикали для стиля DRAW_ARROW	int
PLOT_DRAW_BEGIN	Количество начальных баров без отрисовки и значений в DataWindow	int
PLOT_DRAW_TYPE	Тип графического построения	<a href="#">ENUM_DRAW_TYPE</a>
PLOT_SHOW_DATA	Признак отображения значений построения в окне DataWindow	bool
PLOT_SHIFT	Сдвиг графического построения индикатора по оси времени в барах	int
PLOT_LINE_STYLE	Стиль линии отрисовки	<a href="#">ENUM_LINE_STYLE</a>
PLOT_LINE_WIDTH	Толщина линии отрисовки	int
PLOT_COLOR_INDEXES	Количество цветов	int

PLOT_LINE_COLOR	Цвет отрисовки	color модификатор=номер индекса цвета
-----------------	----------------	---

Для функции [PlotIndexSetDouble\(\)](#)

#### ENUM\_PLOT\_PROPERTY\_DOUBLE

Идентификатор	Описание	Тип свойства
PLOT_EMPTY_VALUE	Пустое значение для построения, для которого нет отрисовки	double

Для функции [PlotIndexSetString\(\)](#)

#### ENUM\_PLOT\_PROPERTY\_STRING

Идентификатор	Описание	Тип свойства
PLOT_LABEL	Имя индикаторной графической серии для отображения в окне DataWindow. Для сложных графических стилей, требующих для отображения несколько индикаторных буферов, имена для каждого буфера можно задать с использованием ";" в качестве разделителя. Пример кода приведен в <a href="#">DRAW_CANDLES</a>	string

Существуют 5 стилей, которыми может рисоваться линия в пользовательском индикаторе. Они применимы только при толщине линии 0 или 1.

#### ENUM\_LINE\_STYLE

Идентификатор	Описание
STYLE_SOLID	Сплошная линия
STYLE_DASH	Прерывистая линия
STYLE_DOT	Пунктирная линия
STYLE_DASHDOT	Штрих-пунктирная линия
STYLE_DASHDOTDOT	Штрих - две точки

Для задания стиля рисования линии и вида отрисовки используется функция [PlotIndexSetInteger\(\)](#). Для расширений Фибоначчи указать толщину и стиль отрисовки уровней можно функцией [ObjectSetInteger\(\)](#).

Пример:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- indicator buffers
double MABuffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- привязка массива к индикаторному буферу с индексом 0
SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
//--- задать рисование линии
PlotIndexSetInteger(0,PLOT_DRAW_TYPE,DRAW_LINE);
//--- задание стиля для рисования линии
PlotIndexSetInteger(0,PLOT_LINE_STYLE,STYLE_DOT);
//--- задание цвета линии
PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrRed);
//--- задание толщины линии
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
//--- задание метки для линии
PlotIndexSetString(0,PLOT_LABEL,"Moving Average");
//---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//---
for(int i=prev_calculated;i<rates_total;i++)
    MABuffer[i]=close[i];
//--- return value of prev_calculated for next call
return(rates_total);
}
```

## Свойства пользовательских индикаторов

Количество индикаторных буферов, которые можно использовать в пользовательском индикаторе, не ограничено. Но каждому массиву, который назначается в качестве индикаторного буфера с помощью функции [SetIndexBuffer\(\)](#), должен быть указан тип данных, которые он будет хранить. Это может быть одно из значений перечисления ENUM\_INDEXBUFFER\_TYPE.

### ENUM\_INDEXBUFFER\_TYPE

Идентификатор	Описание
INDICATOR_DATA	Данные для отрисовки
INDICATOR_COLOR_INDEX	Цвета отрисовки
INDICATOR_CALCULATIONS	Вспомогательные буферы для промежуточных вычислений

Пользовательский индикатор имеет множество настроек для удобства отображения и восприятия. Эти настройки производятся через задание соответствующих свойств индикатора с помощью функций [IndicatorSetDouble\(\)](#), [IndicatorSetInteger\(\)](#) и [IndicatorSetString\(\)](#). Идентификаторы свойств индикатора перечислены в перечислениях ENUM\_CUSTOMIND\_PROPERTY.

### ENUM\_CUSTOMIND\_PROPERTY\_INTEGER

Идентификатор	Описание	Тип свойства
INDICATOR_DIGITS	Точность отображения значений индикатора	int
INDICATOR_HEIGHT	Фиксированная высота собственного окна индикатора (команда препроцессора <a href="#">#property indicator_height</a> )	int
INDICATOR_LEVELS	Количество уровней на окне индикатора	int
INDICATOR_LEVELCOLOR	Цвет линии уровня	color модификатор=номер уровня
INDICATOR_LEVELSTYLE	Стиль линии уровня	<a href="#">ENUM_LINE_STYLE</a> модификатор=номер уровня
INDICATOR_LEVELWIDTH	Толщина линии уровня	int модификатор=номер уровня

### ENUM\_CUSTOMIND\_PROPERTY\_DOUBLE

Идентификатор	Описание	Тип свойства
INDICATOR_MINIMUM	Минимум окна индикатора	double
INDICATOR_MAXIMUM	Максимум окна индикатора	double

INDICATOR_LEVELVALUE	Значение уровня	double модификатор=номер уровня
----------------------	-----------------	------------------------------------

**ENUM\_CUSTOMIND\_PROPERTY\_STRING**

Идентификатор	Описание	Тип свойства
INDICATOR_SHORTNAME	Короткое наименование индикатора	string
INDICATOR_LEVELTEXT	Описание уровня	string модификатор=номер уровня

**Примеры:**

```

//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots 2
#property indicator_type1 DRAW_LINE
#property indicator_type2 DRAW_LINE
#property indicator_color1 clrLightSeaGreen
#property indicator_color2 clrRed
//--- input parameters
extern int KPeriod=5;
extern int DPeriod=3;
extern int Slowing=3;
//--- indicator buffers
double MainBuffer[];
double SignalBuffer[];
double HighesBuffer[];
double LowesBuffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,MainBuffer,INDICATOR_DATA);
SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
SetIndexBuffer(2,HighesBuffer,INDICATOR_CALCULATIONS);
SetIndexBuffer(3,LowesBuffer,INDICATOR_CALCULATIONS);
//--- set accuracy
IndicatorSetInteger(INDICATOR_DIGITS,2);
//--- set levels
IndicatorSetInteger(INDICATOR_LEVELS,2);
IndicatorSetDouble(INDICATOR_LEVELVALUE,0,20);
IndicatorSetDouble(INDICATOR_LEVELVALUE,1,80);
//--- set maximum and minimum for subwindow
IndicatorSetDouble(INDICATOR_MINIMUM,0);

```

```
IndicatorSetDouble(INDICATOR_MAXIMUM, 100);
//--- sets first bar from what index will be drawn
PlotIndexSetInteger(0, PLOT_DRAW_BEGIN, KPeriod+Slowing-2);
PlotIndexSetInteger(1, PLOT_DRAW_BEGIN, KPeriod+Slowing+DPeriod);
//--- set style STYLE_DOT for second line
PlotIndexSetInteger(1, PLOT_LINE_STYLE, STYLE_DOT);
//--- name for DataWindow and indicator subwindow label
IndicatorSetString(INDICATOR_SHORTNAME, "Stoch(\"+KPeriod\"," "+DPeriod\"," "+Slowing+\"");
PlotIndexSetString(0, PLOT_LABEL, "Main");
PlotIndexSetString(1, PLOT_LABEL, "Signal");
//--- sets drawing line to empty value
PlotIndexSetDouble(0, PLOT_EMPTY_VALUE, 0.0);
PlotIndexSetDouble(1, PLOT_EMPTY_VALUE, 0.0);
//--- initialization done
}
```

## Типы технических индикаторов

Существует два способа программно создавать хэндл индикатора для последующего [доступа к его значениям](#). Первый способ состоит в непосредственном указании имени функции из списка [технических индикаторов](#). Второй способ позволяет с помощью функции [IndicatorCreate\(\)](#) единообразно создавать хэндл любого индикатора заданием идентификатора из перечисления `ENUM_INDICATOR`. Оба варианта создания хэндла индикатора равноправны, можно использовать тот, который наиболее удобен в каждом конкретном случае при написании программы на MQL5.

При создании индикатора типа `IND_CUSTOM`, поле `type` первого элемента массива [входных параметров MqlParam](#) обязательно должен иметь значение `TYPE_STRING` из перечисления [ENUM\\_DATATYPE](#), а поле `string_value` первого элемента должно содержать имя пользовательского индикатора.

### `ENUM_INDICATOR`

Идентификатор	Индикатор
<code>IND_AC</code>	Accelerator Oscillator
<code>IND_AD</code>	Accumulation/Distribution
<code>IND_ADX</code>	Average Directional Index
<code>IND_ADXW</code>	ADX by Welles Wilder
<code>IND_ALLIGATOR</code>	Alligator
<code>IND_AMA</code>	Adaptive Moving Average
<code>IND_AO</code>	Awesome Oscillator
<code>IND_ATR</code>	Average True Range
<code>IND_BANDS</code>	Bollinger Bands®
<code>IND_BEARS</code>	Bears Power
<code>IND_BULLS</code>	Bulls Power
<code>IND_BWMFI</code>	Market Facilitation Index
<code>IND_CCI</code>	Commodity Channel Index
<code>IND_CHAIKIN</code>	Chaikin Oscillator
<code>IND_CUSTOM</code>	Custom indicator
<code>IND_DEMA</code>	Double Exponential Moving Average
<code>IND_DEMARKER</code>	DeMarker
<code>IND_ENVELOPES</code>	Envelopes
<code>IND_FORCE</code>	Force Index
<code>IND_FRACTALS</code>	Fractals
<code>IND_FRAMA</code>	Fractal Adaptive Moving Average

IND_GATOR	Gator Oscillator
IND_ICHIMOKU	Ichimoku Kinko Hyo
IND_MA	Moving Average
IND_MACD	MACD
IND_MFI	Money Flow Index
IND_MOMENTUM	Momentum
IND_OBV	On Balance Volume
IND_OSMA	OsMA
IND_RSI	Relative Strength Index
IND_RVI	Relative Vigor Index
IND_SAR	Parabolic SAR
IND_STDDEV	Standard Deviation
IND_STOCHASTIC	Stochastic Oscillator
IND_TEMA	Triple Exponential Moving Average
IND_TRIX	Triple Exponential Moving Averages Oscillator
IND_VIDYA	Variable Index Dynamic Average
IND_VOLUMES	Volumes
IND_WPR	Williams' Percent Range

## Идентификаторы типов данных

При создании хэндла индикатора функцией [IndicatorCreate\(\)](#) необходимо указывать последним параметром массив типа [MqlParam](#). Соответственно, структура [MqlParam](#), описывающая параметры индикатора, содержит специальное поле *type*. Это поле содержит информацию о типе данных ([вещественный](#), [целочисленный](#) или [строковый](#) тип), которые передаются конкретным элементом этого массива. Значение этого поля структуры [MqlParam](#) может быть одним из значений перечисления [ENUM\\_DATATYPE](#).

### ENUM\_DATATYPE

Идентификатор	Тип данных
TYPE_BOOL	bool
TYPE_CHAR	char
TYPE_UCHAR	uchar
TYPE_SHORT	short
TYPE USHORT	ushort
TYPE_COLOR	color
TYPE_INT	int
TYPE_UINT	uint
TYPE_DATETIME	datetime
TYPE_LONG	long
TYPE ULONG	ulong
TYPE_FLOAT	float
TYPE_DOUBLE	double
TYPE_STRING	string

Каждый элемент этого массива описывает соответствующий входной параметр создаваемого [технического индикатора](#), поэтому тип и порядок следования элементов массива должен быть строго выдержан в соответствии с описанием.

## Состояние окружения

Константы, описывающие текущую среду выполнения mq5-программы делятся на группы:

- [Состояние клиентского терминала](#) - информация о клиентском терминале;
- [Информация о запущенной MQL5-программе](#) - свойства mq5-программы, которые помогают дополнительно управлять ее поведением;
- [Информация об инструменте](#) - получение торговой информации об инструменте;
- [Информация о счете](#) - информация о текущем торговом счете;
- [Статистика тестирования](#) - результаты тестирования эксперта.

## Состояние клиентского терминала

Идентификаторы для получения информации о клиентском терминале функциями [TerminalInfoInteger\(\)](#) и [TerminalInfoString\(\)](#). В качестве параметра эти функции принимают значения из перечислений `ENUM_TERMINAL_INFO_INTEGER` и `ENUM_TERMINAL_INFO_STRING` соответственно.

### `ENUM_TERMINAL_INFO_INTEGER`

Идентификатор	Описание	Тип свойства
<code>TERMINAL_BUILD</code>	Номер билда запущенного терминала	<code>int</code>
<code>TERMINAL_COMMUNITY_ACCOUNT</code>	Флаг наличия авторизационных данных MQL5.community в терминале	<code>bool</code>
<code>TERMINAL_COMMUNITY_CONNECTION</code>	Наличие подключения к MQL5.community	<code>bool</code>
<code>TERMINAL_CONNECTED</code>	Наличие подключения к торговому серверу	<code>bool</code>
<code>TERMINAL_DLLS_ALLOWED</code>	Разрешение на использование DLL	<code>bool</code>
<code>TERMINAL_TRADE_ALLOWED</code>	<a href="#">Разрешение на торговлю</a>	<code>bool</code>
<code>TERMINAL_EMAIL_ENABLED</code>	Разрешение на отправку писем с использованием SMTP-сервера и логина, указанных в настройках терминала	<code>bool</code>
<code>TERMINAL_FTP_ENABLED</code>	Разрешение на отправку отчетов по FTP на указанный сервер для указанного в настройках терминала торгового счета	<code>bool</code>
<code>TERMINAL_NOTIFICATIONS_ENABLED</code>	Разрешение на отправку уведомлений на смартфон	<code>bool</code>
<code>TERMINAL_MAXBARS</code>	Максимальное количество баров на графике	<code>int</code>
<code>TERMINAL_MQID</code>	Флаг наличия MetaQuotes ID для отправки <a href="#">Push-уведомлений</a>	<code>bool</code>
<code>TERMINAL_CODEPAGE</code>	Номер <a href="#">кодовой страницы языка</a> , установленного в клиентском терминале	<code>int</code>

TERMINAL_CPU_CORES	Количество процессоров в системе	int
TERMINAL_DISK_SPACE	Объем свободной памяти на диске для папки MQL5\Files терминала (агента), в МБ	int
TERMINAL_MEMORY_PHYSICAL	Размер физической памяти в системе, в МБ	int
TERMINAL_MEMORY_TOTAL	Размер памяти, доступной процессу терминала (агента), в МБ	int
TERMINAL_MEMORY_AVAILABLE	Размер свободной памяти процесса терминала (агента) в МБ	int
TERMINAL_MEMORY_USED	Размер памяти, использованной терминалом (агентом), в МБ	int
TERMINAL_X64	Признак "64 битный терминал"	bool
TERMINAL_OPENCL_SUPPORT	Версия поддерживаемой OpenCL в виде 0x00010002 = 1.2. "0" означает, что OpenCL не поддерживается	int
TERMINAL_SCREEN_DPI	Разрешающая способность вывода информации на экран измеряется в количестве точек на линейный дюйм поверхности (DPI). Знание этого параметра позволяет задавать размеры графических объектов таким образом, чтобы они выглядели одинаково на мониторах с различной разрешающей способностью.	int
TERMINAL_SCREEN_LEFT	Левая координата виртуального экрана. Виртуальным экраном является прямоугольник, охватывающий все мониторы. Если в системе имеется два монитора и их порядок задан справа налево, то левая координата виртуального экрана может оказаться на границе двух мониторов.	int

TERMINAL_SCREEN_TOP	Верхняя координата виртуального экрана	int
TERMINAL_SCREEN_WIDTH	Ширина терминала	int
TERMINAL_SCREEN_HEIGHT	Высота терминала	int
TERMINAL_LEFT	Левая координата терминала относительно виртуального экрана	int
TERMINAL_TOP	Верхняя координата терминала относительно виртуального экрана	int
TERMINAL_RIGHT	Правая координата терминала относительно виртуального экрана	int
TERMINAL_BOTTOM	Нижняя координата терминала относительно виртуального экрана	int
TERMINAL_PING_LAST	Последнее известное значение пинга до торгового сервера в микросекундах. В одной секунде миллион микросекунд	int
TERMINAL_VPS	Признак того, что терминал запущен на виртуальном сервере <a href="#">MetaTrader Virtual Hosting</a> (MetaTrader VPS)	bool
<b>Идентификатор клавиши</b>	<b>Описание</b>	
TERMINAL_KEYSTATE_LEFT	Состояние клавиши "Стрелка влево"	int
TERMINAL_KEYSTATE_UP	Состояние клавиши "Стрелка вверх"	int
TERMINAL_KEYSTATE_RIGHT	Состояние клавиши "Стрелка вправо"	int
TERMINAL_KEYSTATE_DOWN	Состояние клавиши "Стрелка вниз"	int
TERMINAL_KEYSTATE_SHIFT	Состояние клавиши "Shift"	int
TERMINAL_KEYSTATE_CONTROL	Состояние клавиши "Ctrl"	int
TERMINAL_KEYSTATE_MENU	Состояние клавиши "Windows"	int

TERMINAL_KEYSTATE_CAPSLOCK	Состояние клавиши "CapsLock"	int
TERMINAL_KEYSTATE_NUMLOCK	Состояние клавиши "NumLock"	int
TERMINAL_KEYSTATE_SCROLLLOCK	Состояние клавиши "ScrollLock"	int
TERMINAL_KEYSTATE_ENTER	Состояние клавиши "Enter"	int
TERMINAL_KEYSTATE_INSERT	Состояние клавиши "Insert"	int
TERMINAL_KEYSTATE_DELETE	Состояние клавиши "Delete"	int
TERMINAL_KEYSTATE_HOME	Состояние клавиши "Home"	int
TERMINAL_KEYSTATE_END	Состояние клавиши "End"	int
TERMINAL_KEYSTATE_TAB	Состояние клавиши "Tab"	int
TERMINAL_KEYSTATE_PAGEUP	Состояние клавиши "PageUp"	int
TERMINAL_KEYSTATE_PAGEDOWN	Состояние клавиши "PageDown"	int
TERMINAL_KEYSTATE_ESCAPE	Состояние клавиши "Escape"	int

Вызов TerminalInfoInteger(TERMINAL\_KEYSTATE\_XXX) возвращает такой же код состояния клавиши, как и функция [GetKeyState\(\)](#) из MSDN.

**Пример** вычисления коэффициента масштабирования:

```
//--- создаём кнопку шириной 1.5 дюйма на экране
int screen_dpi = TerminalInfoInteger(TERMINAL_SCREEN_DPI); // получим DPI монитора
int base_width = 144; // базовая ширина в экране
int width      = (button_width * screen_dpi) / 96; // вычислим ширину кнопки

...
//--- вычисление коэффициента масштабирования в процентах
int scale_factor=(TerminalInfoInteger(TERMINAL_SCREEN_DPI) * 100) / 96;
//--- использование коэффициента масштабирования
width=(base_width * scale_factor) / 100;
```

При таком использовании графический [ресурс](#) будет иметь одинаковый на глаз размер на мониторах с различной разрешающей способностью. При этом размеры управляющих элементов (кнопки, окна диалогов и т.д.) будут соответствовать настройкам персонализации.

#### ENUM\_TERMINAL\_INFO\_DOUBLE

Идентификатор	Описание	Тип свойства
---------------	----------	--------------

TERMINAL_COMMUNITY_BALANCE	Баланс пользователя в MQL5.community	double
TERMINAL_RETRANSMISSION	<p>Процент повторно отправляемых сетевых пакетов в TCP/IP протоколе для всех запущенных приложений и служб на данном компьютере. Даже в самой быстрой и правильно настроенной сети происходят потери пакетов и, как следствие, отсутствие подтверждений о доставке пакетов между получателем и отправителем. В таких случаях производится повторная отправка "потерянного" пакета.</p> <p>Не является показателем качества подключения конкретного терминала к конкретному торговому серверу, так как считается для всей сетевой активности, включая системную и фоновую.</p> <p>Показатель TERMINAL_RETRANSMISSION запрашивается раз в минуту из операционной системы. Сам терминал не считает этот показатель.</p>	double

[Файловые операции](#) можно проводить только в двух каталогах, пути к которым можно получить при запросе свойств TERMINAL\_DATA\_PATH и TERMINAL\_COMMONDATA\_PATH.

#### ENUM\_TERMINAL\_INFO\_STRING

Идентификатор	Описание	Тип свойства
TERMINAL_LANGUAGE	Язык терминала	string
TERMINAL_COMPANY	Имя компании	string
TERMINAL_NAME	Имя терминала	string
TERMINAL_PATH	Папка, из которой запущен терминал	string

TERMINAL_DATA_PATH	Папка, в которой хранятся данные терминала	string
TERMINAL_COMMONDATA_PATH	Общая папка всех клиентских терминалов, установленных на компьютере	string

Для лучшего понимания путей, хранящихся в свойствах параметров TERMINAL\_PATH, TERMINAL\_DATA\_PATH и TERMINAL\_COMMONDATA\_PATH, рекомендуется выполнить скрипт, которых сообщит эти значения для данной копии терминала, установленного на вашем компьютере.

#### Пример: скрипт выводит информацию о путях терминала

```
//+-----+
//| Check_TerminalPaths.mq5 |
//| Copyright 2009, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//---
Print("TERMINAL_PATH = ",TerminalInfoString(TERMINAL_PATH));
Print("TERMINAL_DATA_PATH = ",TerminalInfoString(TERMINAL_DATA_PATH));
Print("TERMINAL_COMMONDATA_PATH = ",TerminalInfoString(TERMINAL_COMMONDATA_PATH));
}
```

В результате выполнения скрипта в Журнал Экспертов будут выведены сообщения, подобные приведенным на рисунке ниже.

Инструменты		
Время	Источник	Сообщение
2018.06.28 16:49:25.234	Paths (EURUSD,H1)	TERMINAL_PATH = C:\Program Files\MetaTrader 5
2018.06.28 16:49:25.234	Paths (EURUSD,H1)	TERMINAL_DATA_PATH = C:\Users\smith\AppData\Roaming\MetaQuotes\T...
2018.06.28 16:49:25.234	Paths (EURUSD,H1)	TERMINAL_COMMONDATA_PATH = C:\Users\smith\AppData\Roaming\MetaQu...

## Информация о запущенной MQL5-программе

Константы, для получения информации о выполняющейся mql5-программе, перечислены в ENUM\_MQL\_INFO\_INTEGER и ENUM\_MQL\_INFO\_STRING.

Для функции [MQLInfoInteger\(\)](#)

### ENUM\_MQL\_INFO\_INTEGER

Идентификатор	Описание	Тип свойства
MQL_MEMORY_LIMIT	Максимально возможный объём динамической памяти для MQL5-программы в МБ	int
MQL_MEMORY_USED	Размер использованной памяти MQL5-программой в МБ	int
MQL_PROGRAM_TYPE	Тип mql5-программы	<a href="#">ENUM_PROGRAM_TYPE</a>
MQL_DLLS_ALLOWED	Разрешение на использование DLL для данной запущенной программы	bool
MQL_TRADE_ALLOWED	<a href="#">Разрешение на торговлю</a> для данной запущенной программы	bool
MQL_SIGNALS_ALLOWED	Разрешение на работу с сигналами данной запущенной программы	bool
MQL_DEBUG	Признак работы запущенной программы в режиме отладки	bool
MQL_PROFILER	Признак работы запущенной программы в режиме профилирования кода	bool
MQL_TESTER	Признак работы запущенной программы в тестере	bool
MQL_FORWARD	Признак работы запущенной программы в процессе форвардного тестирования	bool
MQL_OPTIMIZATION	Признак работы запущенной программы в процессе оптимизации	bool
MQL_VISUAL_MODE	Признак работы запущенной программы в визуальном режиме тестирования	bool

MQL_FRAME_MODE	Признак работы запущенного эксперта на графике в режиме сбора фреймов результатов оптимизации	bool
MQL_LICENSE_TYPE	Тип лицензии модуля EX5. Лицензия относится именно к тому модулю EX5, из которого делается запрос с помощью MQLInfoInteger(MQL_LICENSE_TYPE).	<a href="#">ENUM_LICENSE_TYPE</a>

Для функции [MQLInfoString\(\)](#)

#### ENUM\_MQL\_INFO\_STRING

Идентификатор	Описание	Тип свойства
MQL_PROGRAM_NAME	Имя запущенной MQL5-программы	string
MQL_PROGRAM_PATH	Путь для данной запущенной программы	string

Для получения информации о типе выполняемой программы предназначены значения перечисления ENUM\_PROGRAM\_TYPE.

#### ENUM\_PROGRAM\_TYPE

Идентификатор	Описание
PROGRAM_SCRIPT	Скрипт
PROGRAM_EXPERT	Эксперт
PROGRAM_INDICATOR	Индикатор

#### ENUM\_LICENSE\_TYPE

Идентификатор	Описание
LICENSE_FREE	Бесплатная неограниченная версия
LICENSE_DEMO	Демо-версия платного продукта из Маркета. Работает только в тестере стратегий
LICENSE_FULL	Купленная лицензионная версия допускает не менее 5 активаций. Продавец может увеличить разрешенное число активаций
LICENSE_TIME	Версия с ограниченной по времени лицензией

Пример:

```
ENUM_PROGRAM_TYPE mql_program=(ENUM_PROGRAM_TYPE)MQLInfoInteger(MQL_PROGRAM_TYPE);
switch(mql_program)
{
    case PROGRAM_SCRIPT:
    {
        Print(__FILE__+" is script");
        break;
    }
    case PROGRAM_EXPERT:
    {
        Print(__FILE__+" is Expert Advisor");
        break;
    }
    case PROGRAM_INDICATOR:
    {
        Print(__FILE__+" is custom indicator");
        break;
    }
    default:Print("MQL5 type value is ",mql_program);
}
```

## Информация об инструменте

Для получения текущей рыночной информации служат функции [SymbolInfoInteger\(\)](#), [SymbolInfoDouble\(\)](#) и [SymbolInfoString\(\)](#). В качестве второго параметра этих функций допустимо передавать один из идентификаторов из перечислений ENUM\_SYMBOL\_INFO\_INTEGER, ENUM\_SYMBOL\_INFO\_DOUBLE и ENUM\_SYMBOL\_INFO\_STRING соответственно.

Для функции [SymbolInfoInteger\(\)](#)

### ENUM\_SYMBOL\_INFO\_INTEGER

Идентификатор	Описание	Тип свойства
SYMBOL_CUSTOM	Признак того, что символ является пользовательским, то есть искусственно созданным на основе других символов из Market Watch или/и внешних источников данных	bool
SYMBOL_BACKGROUND_COLOR	Цвет фона, которым подсвечивается символ в Market Watch	color
SYMBOL_CHART_MODE	Тип цены для построения баров - Bid или Last	<a href="#">ENUM_SYMBOL_CHART_MODE</a>
SYMBOL_EXIST	Признак того, что символ с таким именем существует	bool
SYMBOL_SELECT	Признак того, что символ выбран в Market Watch	bool
SYMBOL_VISIBLE	Признак того, что выбранный символ отображается в Market Watch.  Некоторые символы (как правило, это кросс-курсы, которые необходимы для расчёта маржевых требований и прибыли в валюте депозита) выбираются автоматически, при этом могут не отображаться в Market Watch. Для отображения такие символы должны быть выбраны явно.	bool
SYMBOL_SESSION DEALS	Количество сделок в текущей сессии	long

SYMBOL_SESSION_BUY_ORDERS	Общее число ордеров на покупку в текущий момент	long
SYMBOL_SESSION_SELL_ORDERS	Общее число ордеров на продажу в текущий момент	long
SYMBOL_VOLUME	Volume - объем в последней сделке	long
SYMBOL_VOLUMEHIGH	Максимальный Volume за день	long
SYMBOL_VOLUMELOW	Минимальный Volume за день	long
SYMBOL_TIME	Время последней котировки	datetime
SYMBOL_DIGITS	Количество знаков после запятой	int
SYMBOL_SPREAD	Размер спреда в пунктах	int
SYMBOL_SPREAD_FLOAT	Признак плавающего спреда	bool
SYMBOL_TICKS_BOOKDEPTH	Максимальное количество показываемых заявок в <u>стакане</u> . Для инструментов, не имеющих очереди заявок, значение равно 0	int
SYMBOL_TRADE_CALC_MODE	Способ вычисления стоимости контракта	<a href="#">ENUM_SYMBOL_CALC_MODE</a>
SYMBOL_TRADE_MODE	Тип исполнения ордеров	<a href="#">ENUM_SYMBOL_TRADE_MODE</a>
SYMBOL_START_TIME	Дата начала торгов по инструменту (обычно используется для фьючерсов)	datetime
SYMBOL_EXPIRATION_TIME	Дата окончания торгов по инструменту (обычно используется для фьючерсов)	datetime
SYMBOL_TRADE_STOPS_LEVEL	Минимальный отступ в пунктах от текущей цены закрытия для установки Stop ордеров	int
SYMBOL_TRADE_FREEZE_LEVEL	Дистанция заморозки торговых операций (в пунктах)	int
SYMBOL_TRADE_EXEMODE	Режим заключения сделок	<a href="#">ENUM_SYMBOL_TRADE_EXECUTION</a>
SYMBOL_SWAP_MODE	Модель расчета свопа	<a href="#">ENUM_SYMBOL_SWAP_MODE</a>
SYMBOL_SWAP_ROLLOVER3DAYS	День недели для начисления тройного свопа	<a href="#">ENUM_DAY_OF_WEEK</a>

SYMBOL_MARGIN_HEDGED_USE_LEG	Режим расчета хеджированной маржи по наибольшей стороне (Buy или Sell)	bool
SYMBOL_EXPIRATION_MODE	Флаги разрешенных <a href="#">режимов истечения</a> ордера	int
SYMBOL_FILLING_MODE	Флаги разрешенных <a href="#">режимов заливки</a> ордера	int
SYMBOL_ORDER_MODE	Флаги разрешенных <a href="#">типов ордера</a>	int
SYMBOL_ORDER_GTC_MODE	Срок действия StopLoss и TakeProfit ордеров, если SYMBOL_EXPIRATION_MODE= <a href="#">SYMBOL_EXPIRATION_GTC</a> (Good till cancelled)	<a href="#">ENUM_SYMBOL_ORDER_GTC_MODE</a>
SYMBOL_OPTION_MODE	Тип опциона	<a href="#">ENUM_SYMBOL_OPTION_MODE</a>
SYMBOL_OPTION_RIGHT	Право опциона (Call/Put)	<a href="#">ENUM_SYMBOL_OPTION_RIGHT</a>

Для функции [SymbolInfoDouble\(\)](#)

#### ENUM\_SYMBOL\_INFO\_DOUBLE

Идентификатор	Описание	Тип свойства
SYMBOL_BID	Bid - лучшее предложение на продажу	double
SYMBOL_BIDHIGH	Максимальный Bid за день	double
SYMBOL_BIDLOW	Минимальный Bid за день	double
SYMBOL_ASK	Ask - лучшее предложение на покупку	double
SYMBOL_ASKHIGH	Максимальный Ask за день	double
SYMBOL_ASKLOW	Минимальный Ask за день	double
SYMBOL_LAST	Цена, по которой совершена последняя сделка	double
SYMBOL_LASTHIGH	Максимальный Last за день	double
SYMBOL_LASTLOW	Минимальный Last за день	double
SYMBOL_VOLUME_REAL	Volume за день	double
SYMBOL_VOLUMEHIGH_REAL	Максимальный Volume за день	double
SYMBOL_VOLUMELOW_REAL	Минимальный Volume за день	double

SYMBOLOPTION_STRIKE	Цена исполнения опциона. Это цена, по которой покупатель опциона может купить (при опционе Call) или продать (при опционе Put) базовый актив, а продавец опциона соответственно обязан продать или купить соответствующее количество базового актива.	double
SYMBOLPOINT	Значение одного пункта	double
SYMBOLTRADE_TICK_VALUE_P ROFIT	Значение SYMBOLTRADE_TICK_VALUE_P ROFIT	double
SYMBOLTRADE_TICK_VALUE_ PROFIT	Рассчитанная стоимость тика для прибыльной позиции	double
SYMBOLTRADE_TICK_VALUE_ LOSS	Рассчитанная стоимость тика для убыточной позиции	double
SYMBOLTRADE_TICK_SIZE	Минимальное изменение цены	double
SYMBOLTRADE_CONTRACT_SI ZE	Размер торгового контракта	double
SYMBOLTRADE_ACCRUED_IN TEREST	<u>Накопленный купонный доход</u> - часть купонного процентного дохода по облигации, рассчитываемая пропорционально количеству дней, прошедших от даты выпуска купонной облигации или даты выплаты предшествующего купонного дохода	double
SYMBOLTRADE_FACE_VALUE	<u>Номинальная стоимость</u> - начальная стоимость облигации, установленная эмитентом	double
SYMBOLTRADE_LIQUIDITY_RA TE	Коэффициент ликвидности - доля от стоимости актива, которую можно использовать в качестве залога.	double
SYMBOLVOLUME_MIN	Минимальный объем для заключения сделки	double
SYMBOLVOLUME_MAX	Максимальный объем для заключения сделки	double

SYMBOL_VOLUME_STEP	Минимальный шаг изменения объема для заключения сделки	double
SYMBOL_VOLUME_LIMIT	Максимально допустимый для данного символа совокупный объем открытой позиции и отложенных ордеров в одном направлении (покупка или продажа). Например, при ограничении в 5 лотов можно иметь открытую позицию на покупку объемом 5 лотов и выставить отложенный ордер Sell Limit объемом 5 лотов. Но при этом нельзя выставить отложенный ордер Buy Limit (поскольку совокупный объем в одном направлении превысит ограничение) или выставить Sell Limit объемом более 5 лотов.	double
SYMBOL_SWAP_LONG	Значение свопа в покупку	double
SYMBOL_SWAP_SHORT	Значение свопа в продажу	double
SYMBOL_MARGIN_INITIAL	Начальная (иницирующая) маржа обозначает размер необходимых залоговых средств в маржинальной валюте для открытия позиции объемом в один лот. Используется при проверке средств клиента при входе в рынок.	double
SYMBOL_MARGIN_MAINTENANCE	Поддерживающая маржа по инструменту. В случае если задана - указывает размер маржи в маржинальной валюте инструмента, поддерживаемой с одного лота. Используется при проверке средств клиента при изменении состояния счета клиента. Если поддерживающая маржа равна 0, то используется начальная маржа.	double
SYMBOL_SESSION_VOLUME	Суммарный объем сделок в текущую сессию	double

SYMBOL_SESSION_TURNOVER	Суммарный оборот в текущую сессию	double
SYMBOL_SESSION_INTEREST	Суммарный объём открытых позиций	double
SYMBOL_SESSION_BUY_ORDER_S_VOLUME	Общий объём ордеров на покупку в текущий момент	double
SYMBOL_SESSION_SELL_ORDERS_VOLUME	Общий объём ордеров на продажу в текущий момент	double
SYMBOL_SESSION_OPEN	Цена открытия сессии	double
SYMBOL_SESSION_CLOSE	Цена закрытия сессии	double
SYMBOL_SESSION_AW	Средневзвешенная цена сессии	double
SYMBOL_SESSION_PRICE_SETTLEMENT	Цена поставки на текущую сессию	double
SYMBOL_SESSION_PRICE_LIMIT_MIN	Минимально допустимое значение цены на сессию	double
SYMBOL_SESSION_PRICE_LIMIT_MAX	Максимально допустимое значение цены на сессию	double
SYMBOL_MARGIN_HEDGED	<p>Размер контракта или маржи для одного лота перекрытых позиций (разноравленные позиции по одному символу). Существует два способа расчета маржи для перекрытых позиций. Способ расчета определяется брокером.</p> <p>Базовый расчет:</p> <ul style="list-style-type: none"> <li>Если для инструмента задана первоначальная маржа (SYMBOL_MARGIN_INITIAL), то хеджированная маржа указывается как абсолютное значение (в деньгах).</li> <li>Если первоначальная маржа не задана (равна 0), то в SYMBOL_MARGIN_HEDGED указывается размер контракта, который будет использован при расчете маржи по формуле,</li> </ul>	double

	<p>соответствующей типу торгового инструмента (<a href="#">SYMBOL_TRADE_CALC_MODE</a>).</p> <p>Расчет по наибольшей позиции:</p> <ul style="list-style-type: none"> <li>• Значение <code>SYMBOL_MARGIN_HEDGED</code> не учитывается.</li> <li>• Вычисляется объем всех коротких и всех длинных позиций по инструменту.</li> <li>• Для каждой стороны рассчитывается средневзвешенная цена открытия, а также средневзвешенная цена конвертации в валюту депозита.</li> <li>• Далее по формулам, соответствующим типу инструмента (<a href="#">SYMBOL_TRADE_CALC_MODE</a>), рассчитывается маржа для короткой и для длинной стороны.</li> <li>• В качестве итогового значения используется наибольшее.</li> </ul>	
--	---	--

Для функции [SymbolInfoString\(\)](#)

#### ENUM\_SYMBOL\_INFO\_STRING

Идентификатор	Описание	Тип свойства
<code>SYMBOL_BASIS</code>	Имя базового актива для производного инструмента	<code>string</code>
<code>SYMBOL_CATEGORY</code>	Название категории или сектора, к которой принадлежит торговый символ	<code>string</code>
<code>SYMBOL_CURRENCY_BASE</code>	Базовая валюта инструмента	<code>string</code>
<code>SYMBOL_CURRENCY_PROFIT</code>	Валюта прибыли	<code>string</code>
<code>SYMBOL_CURRENCY_MARGIN</code>	Валюта, в которой вычисляются залоговые средства	<code>string</code>
<code>SYMBOL_BANK</code>	Источник текущей котировки	<code>string</code>

SYMBOL_DESCRIPTION	Строковое описание символа	string
SYMBOL_EXCHANGE	Название биржи или торговой площадки, на которой торгуется символ	string
SYMBOL_FORMULA	<p>Формула для построения цены пользователя символа. Если имя финансового инструмента , входящего в формулу, начинается с цифры или содержит спецсимвол (" ", ".", "-", "&amp;", "#" и т.д.), то имя этого инструмента должно быть заключено в кавычки.</p> <ul style="list-style-type: none"> <li>• Синтетический символ: "@ESU19"/EURCAD</li> <li>• Календарный спред: "Si-9.13"- "Si-6.13"</li> <li>• Индекс евро: 34.38805726 pow(EURUSD,0.3155) pow(EURGBP,0.3056) pow(EURJPY,0.1891) pow(EURCHF,0.1113) pow(EURSEK,0.0785)</li> </ul>	string
SYMBOL_ISIN	Имя торгового символа в системе международных идентификационных кодов ценных бумаг – ISIN (International Securities Identification Number). Международный идентификационный код ценной бумаги – это 12-разрядный буквенно-цифровой код, однозначно идентифицирующий ценную бумагу. Наличие данного свойства символа определяется на стороне торгового сервера.	string
SYMBOL_PAGE	Адрес интернет страницы с информацией по символу. Данный адрес будет отображаться в виде ссылки при просмотре свойств символа в терминале	string
SYMBOL_PATH	Путь в дереве символов	string

Ценовой график по символу может строиться на основе цены Bid или Last. От выбора цены для построения графиков зависит то, как формируются и отображаются бары в терминале. Возможные значения свойства SYMBOL\_CHART\_MODE приводятся в перечислении ENUM\_SYMBOL\_CHART\_MODE

## ENUM\_SYMBOL\_CHART\_MODE

Идентификатор	Описание
SYMBOL_CHART_MODE_BID	Бары строятся по ценам Bid
SYMBOL_CHART_MODE_LAST	Бары строятся по ценам Last

Для каждого финансового инструмента могут быть указаны несколько режимов срока действия (истечения) отложенных ордеров. Каждому режиму сопоставлен флаг, флаги могут комбинироваться операцией логического ИЛИ (`|`), например, `SYMBOL_EXPIRATION_GTC | SYMBOL_EXPIRATION_SPECIFIED`. Чтобы проверить разрешенность конкретного режима для инструмента, необходимо результат логического И (`&`) сравнить с флагом режима.

Если для символа указан флаг `SYMBOL_EXPIRATION_SPECIFIED`, то при отправке отложенного ордера можно конкретно указать, до какого момента действует данный отложенный ордер.

Идентификатор	Значение	Описание
SYMBOL_EXPIRATION_GTC	1	Ордер действителен неограниченно по времени до явной его отмены
SYMBOL_EXPIRATION_DAY	2	Ордер действителен до конца дня
SYMBOL_EXPIRATION_SPECIFIED	4	Срок истечения указывается в ордере
SYMBOL_EXPIRATION_SPECIFIED_DAY	8	День истечения указывается в ордере

## Пример:

```
//+-----+
//| проверяет разрешенность указанного режима экспирации |
//+-----+
bool IsExpirationTypeAllowed(string symbol,int exp_type)
{
//--- получим значение свойства, описывающего допустимые режимы истечения срока действия
int expiration=(int)SymbolInfoInteger(symbol,SYMBOL_EXPIRATION_MODE);
//--- вернем true, если режим exp_type разрешен
return((expiration&exp_type)==exp_type);
}
```

Если свойство `SYMBOL_EXPIRATION_MODE` имеет значение `SYMBOL_EXPIRATION_GTC` (ордер действителен до отмены), то срок действия отложенных ордеров и установленных уровней `StopLoss/TakeProfit` дополнительно задается с помощью перечисления `ENUM_SYMBOL_ORDER_GTC_MODE`.

## ENUM\_SYMBOL\_ORDER\_GTC\_MODE

Идентификатор	Описание
SYMBOL_ORDERS_GTC	Отложенные ордера и уровни Stop Loss/Take Profit действительны неограниченно по времени до явной отмены
SYMBOL_ORDERS_DAILY	Ордера действуют только внутри одного торгового дня. По его завершении все уровни StopLoss и TakeProfit удаляются, а также удаляются отложенные ордера
SYMBOL_ORDERS_DAILY_EXCLUDING_STOPS	При смене торгового дня удаляются только отложенные ордера, уровни StopLoss и TakeProfit сохраняются

При отправке ордера можно указать политику заполнения заявленного в торговом приказе объема. Допустимые варианты исполнения ордера по объему для каждого символа указаны таблице. Для каждого инструмента может быть установлен не один режим, а несколько через комбинацию флагов. Комбинация флагов выражается операцией логического ИЛИ (||), например, SYMBOL\_FILLING\_FOK|SYMBOL\_FILLING\_IOC. Чтобы проверить разрешенность конкретного режима для инструмента, необходимо результат логического И (&) сравнить с флагом режима.

Политика заполнения	Идентификатор	Значение	Описание
Все/Ничего	SYMBOL_FILLING_FOK	1	Данная политика исполнения означает, что ордер может быть выполнен исключительно в указанном объеме. Если на рынке в данный момент не присутствует достаточного объема финансового инструмента, то ордер не будет выполнен. Необходимый объем может быть составлен из нескольких предложений, доступных в данный момент на рынке.
Все/Частично	SYMBOL_FILLING_IOC	2	В данном случае трейдер соглашается

			совершить сделку по максимально доступному на рынке объему в пределах указанного в ордере. В случае невозможности полного исполнения ордер будет выполнен на доступный объем, а неисполненный объем ордера будет отменен. Возможность использования IOC ордеров определяется на торговом сервере.
Вернуть	Идентификатор отсутствует		Данный режим используется для рыночных (Buy и Sell), лимитных и стоп-лимитных ордеров и только в режимах "Исполнение по рынку" и "Биржевое исполнение". В случае частичного исполнения рыночный или лимитный ордер с остаточным объемом не снимается, а продолжает действовать.

В [режимах исполнения](#) "По запросу" и "Немедленный" для рыночных ордеров всегда используется политика заполнения Все/Ничего, а для лимитных ордеров - режим "Вернуть". В данном случае, при отсылке ордеров функциями [OrderSend](#) или [OrderSendAsync](#) тип заполнения для них можно не указывать.

В режимах исполнения "По рынку" и "Биржевой" политика заполнения "Вернуть" всегда разрешена для всех типов ордеров. Разрешенность остальных типов проверяется при помощи свойств [SYMBOL\\_FILLING\\_FOK](#) и [SYMBOL\\_FILLING\\_IOC](#).

#### Пример:

```
//+-----+
//| проверяет разрешенность указанного режима заполнения |
```

```
//+
bool IsFillingTypeAllowed(string symbol,int fill_type)
{
//--- получим значение свойства, описывающего режим заполнения
int filling=(int)SymbolInfoInteger(symbol,SYMBOL_FILLING_MODE);
//--- вернем true, если режим fill_type разрешен
return((filling&fill_type)==fill_type);
}
```

При отправке [торгового запроса](#) функцией OrderSend() для некоторых операций необходимо указать тип ордера из [перечисления ENUM\\_ORDER\\_TYPE](#). Не все типы ордеров могут быть разрешены для конкретного финансового инструмента, свойство [SYMBOL\\_ORDER\\_MODE](#) описывает флаги разрешенных типов ордеров.

Идентификатор	Значение	Описание
SYMBOL_ORDER_MARKET	1	Разрешены рыночные ордера (Buy и Sell)
SYMBOL_ORDER_LIMIT	2	Разрешены лимитные ордера (Buy Limit и Sell Limit)
SYMBOL_ORDER_STOP	4	Разрешены стоп-ордера (Buy Stop и Sell Stop)
SYMBOL_ORDER_STOP_LIMIT	8	Разрешены стоп-лимит ордера (Buy Stop Limit и Sell Stop Limit)
SYMBOL_ORDER_SL	16	Разрешена установка Stop Loss
SYMBOL_ORDER_TP	32	Разрешена установка Take Profit
SYMBOL_ORDER_CLOSEBY	64	Разрешение на закрытие позиции с помощью встречной - то есть открытой в противоположном направлении по тому же инструменту. Свойство задается для счетов с хеджинговой системой учета ( <a href="#">ACCOUNT_MARGIN_MODE_R</a> <a href="#">ETAIL_HEDGING</a> )

**Пример:**

```
//+
//| Функция выводит на печать разрешенные для символа типы ордеров |
//+
void Check_SYMBOL_ORDER_MODE(string symbol)
```

```

{
//--- получим значение свойства, описывающего разрешенные типы ордеров
int symbol_order_mode=(int)SymbolInfoInteger(symbol,SYMBOL_ORDER_MODE);
//--- проверка на рыночные ордера (Market Execution)
if((SYMBOL_ORDER_MARKET&symbol_order_mode)==SYMBOL_ORDER_MARKET)
    Print(symbol+": Рыночные ордера разрешены (Buy и Sell)");
//--- проверка на ордера типа Limit
if((SYMBOL_ORDER_LIMIT&symbol_order_mode)==SYMBOL_ORDER_LIMIT)
    Print(symbol+": Ордера Buy Limit и Sell Limit разрешены");
//--- проверка на ордера типа Stop
if((SYMBOL_ORDER_STOP&symbol_order_mode)==SYMBOL_ORDER_STOP)
    Print(symbol+": Ордера Buy Stop и Sell Stop разрешены");
//--- проверка на ордера типа Stop Limit
if((SYMBOL_ORDER_STOP_LIMIT&symbol_order_mode)==SYMBOL_ORDER_STOP_LIMIT)
    Print(symbol+": Ордера Buy Stop Limit и Sell Stop Limit разрешены");
//--- проверка на возможность установки ордеров Stop Loss
if((SYMBOL_ORDER_SL&symbol_order_mode)==SYMBOL_ORDER_SL)
    Print(symbol+": Ордера Stop Loss разрешены");
//--- проверка на возможность установки ордеров Take Profit
if((SYMBOL_ORDER_TP&symbol_order_mode)==SYMBOL_ORDER_TP)
    Print(symbol+": Ордера Take Profit разрешены");
//---
}

```

Для получения информации о способе вычисления величины залоговых средств по инструменту (размера маржинальных требований) предназначено перечисление ENUM\_SYMBOL\_CALC\_MODE.

#### ENUM\_SYMBOL\_CALC\_MODE

Идентификатор	Описание	Формула
SYMBOL_CALC_MODE_FOREX	Forex mode - расчет прибыли и маржи для Форекс	Margin: Lots * <u>Contract_Size</u> / <u>Leverage</u> * <u>Margin_Rate</u> Profit: (close_price - open_price) * Contract_Size*Lots
SYMBOL_CALC_MODE_FOREX_NO_LEVERAGE	Forex No Leverage mode - расчет прибыли и маржи для Форекс без учета плеча	Margin: Lots * Contract_Size * Margin_Rate Profit: (close_price - open_price) * Contract_Size * Lots
SYMBOL_CALC_MODE_FUTURES	Futures mode - расчет залога и прибыли для фьючерсов	Margin: Lots * InitialMargin * Margin_Rate

		Profit: $(\text{close\_price} - \text{open\_price}) * \text{TickPrice} / \text{TickSize} * \text{Lots}$
SYMBOL_CALC_MODE_CFD	CFD mode - расчет залога и прибыли для CFD	Margin: $\text{Lots} * \text{ContractSize} * \text{MarketPrice} * \text{Margin\_Rate}$  Profit: $(\text{close\_price} - \text{open\_price}) * \text{Contract\_Size} * \text{Lots}$
SYMBOL_CALC_MODE_CFDINDEX	CFD index mode - расчет залога и прибыли для CFD на индексы	Margin: $(\text{Lots} * \text{ContractSize} * \text{MarketPrice}) * \text{TickPrice} / \text{TickSize} * \text{Margin\_Rate}$  Profit: $(\text{close\_price} - \text{open\_price}) * \text{Contract\_Size} * \text{Lots}$
SYMBOL_CALC_MODE_CFDLEVERAGE	CFD Leverage mode - расчет залога и прибыли для CFD при торговле с плечом	Margin: $(\text{Lots} * \text{ContractSize} * \text{MarketPrice}) / \text{Leverage} * \text{Margin\_Rate}$  Profit: $(\text{close\_price} - \text{open\_price}) * \text{Contract\_Size} * \text{Lots}$
SYMBOL_CALC_MODE_EXCH_STOCKS	Exchange mode - расчет залога и прибыли для торговли ценными бумагами на бирже	Margin: $\text{Lots} * \text{ContractSize} * \text{LastPrice} * \text{Margin\_Rate}$  Profit: $(\text{close\_price} - \text{open\_price}) * \text{Contract\_Size} * \text{Lots}$
SYMBOL_CALC_MODE_EXCH_FUTURES	Futures mode - расчет залога и прибыли для торговли фьючерсными контрактами на бирже	Margin: $\text{Lots} * \text{InitialMargin} * \text{Margin\_Rate}$ или $\text{Lots} * \text{MaintenanceMargin} * \text{Margin\_Rate}$  Profit: $(\text{close\_price} - \text{open\_price}) * \text{Lots} * \text{TickPrice} / \text{TickSize}$
SYMBOL_CALC_MODE_EXCH_FUTURES_FORTS	FORTS Futures mode - расчет залога и прибыли для торговли фьючерсными контрактами на FORTS. Размер маржи может уменьшаться на величину отклонения MarginDiscount по следующим правилам: 1. Если цена длинной позиции (ордера на покупку) меньше расчетной цены, то	Margin: $\text{Lots} * \text{InitialMargin} * \text{Margin\_Rate}$ или $\text{Lots} * \text{MaintenanceMargin} * \text{Margin\_Rate} * \text{Margin\_Rate}$  Profit: $(\text{close\_price} - \text{open\_price}) * \text{Lots} * \text{TickPrice} / \text{TickSize}$

	<p>MarginDiscount =  <math>\text{Lots} * ((\text{PriceSettle} - \text{PriceOrder}) * \text{TickPrice} / \text{TickSize})</math>          2. Если цена короткой позиций (ордера на продажу) больше расчетной цены, то  <math>\text{MarginDiscount} = \text{Lots} * ((\text{PriceOrder} - \text{PriceSettle}) * \text{TickPrice} / \text{TickSize})</math>          где:</p> <ul style="list-style-type: none"> <li>○ PriceSettle - расчетная (клиринговая) цена предыдущей сессии;</li> <li>○ PriceOrder - средневзвешенная цена позиции или цена открытия, указанная в ордере (заявке);</li> <li>○ TickPrice - цена тика (стоимость изменения цены на один пункт);</li> <li>○ TickSize - размер тика (минимальный шаг изменения цены)</li> </ul>	
SYMBOL_CALC_MODE_EXCH_BONDS	Exchange Bonds mode - расчет залога и прибыли для торговли облигациями на бирже	Margin: $\text{Lots} * \text{ContractSize} * \text{FaceValue} * \text{open\_price} / 100$ Profit: $\text{Lots} * \text{close\_price} * \text{FaceValue} * \text{Contract\_Size} + \text{AccruedInterest} * \text{Lots} * \text{ContractSize}$
SYMBOL_CALC_MODE_EXCH_STOCKS_MOEX	Exchange MOEX Stocks mode - расчет залога и прибыли для торговли ценными бумагами на бирже МОEX	Margin: $\text{Lots} * \text{ContractSize} * \text{LastPrice} * \text{Margin\_Rate}$ Profit: $(\text{close\_price} - \text{open\_price}) * \text{Contract\_Size} * \text{Lots}$
SYMBOL_CALC_MODE_EXCH_BONDS_MOEX	Exchange MOEX Bonds mode - расчет залога и прибыли для торговли облигациями на бирже МОEX	Margin: $\text{Lots} * \text{ContractSize} * \text{FaceValue} * \text{open\_price} / 100$ Profit: $\text{Lots} * \text{close\_price} * \text{FaceValue} * \text{Contract\_Size} + \text{AccruedInterest} * \text{Lots} * \text{ContractSize}$
SYMBOL_CALC_MODE_SERV_COLLATERAL	Collateral mode - инструмент используется в качестве неторгуемого актива на торговом счете. Рыночная	Margin: нет Profit: нет

	стоимость открытой позиции рассчитывается на основании объема, текущей цены рынка, размера контракта и коэффициента ликвидности. Стоимость учитывается в Активах (Assets), которые суммируются с собственными средствами (Equity). Тем самым открытые позиции по такому инструменту увеличивают размер свободных средств (Free Margin) и служат дополнительным обеспечением под открытые позиции по торгуемым инструментам	Рыночная стоимость: Lots * ContractSize * MarketPrice * LiquidityRate
--	--	---

Существует несколько режимов торговли по финансовым инструментам. Информация о режимах торговли по конкретному инструменту отображена в значениях перечисления ENUM\_SYMBOL\_TRADE\_MODE.

#### ENUM\_SYMBOL\_TRADE\_MODE

Идентификатор	Описание
SYMBOL_TRADE_MODE_DISABLED	Торговля по символу запрещена
SYMBOL_TRADE_MODE_LONGONLY	Разрешены только покупки
SYMBOL_TRADE_MODE_SHORTONLY	Разрешены только продажи
SYMBOL_TRADE_MODE_CLOSEONLY	Разрешены только операции закрытия позиций
SYMBOL_TRADE_MODE_FULL	Нет ограничений на торговые операции

Возможные режимы заключения сделок по конкретному инструменту определены в перечислении ENUM\_SYMBOL\_TRADE\_EXECUTION.

#### ENUM\_SYMBOL\_TRADE\_EXECUTION

Идентификатор	Описание
SYMBOL_TRADE_EXECUTION_REQUEST	Торговля по запросу
SYMBOL_TRADE_EXECUTION_INSTANT	Торговля по потоковым ценам
SYMBOL_TRADE_EXECUTION_MARKET	Исполнение ордеров по рынку
SYMBOL_TRADE_EXECUTION_EXCHANGE	Биржевое исполнение

Способы начисления свопов при переносе позиции указаны в перечислении ENUM\_SYMBOL\_SWAP\_MODE. Способ начисления свопов определяет единицы измерения параметров [SYMBOL\\_SWAP\\_LONG](#) и [SYMBOL\\_SWAP\\_SHORT](#). Например, если свопы начисляются в валюте депозита клиента, в параметрах объем начисляемых свопов указывается именно в валюте депозита клиента.

#### ENUM\_SYMBOL\_SWAP\_MODE

Идентификатор	Описание
SYMBOL_SWAP_MODE_DISABLED	Нет свопов
SYMBOL_SWAP_MODE_POINTS	Свопы начисляются в пунктах
SYMBOL_SWAP_MODE_CURRENCY_SYMBOL	Свопы начисляются в деньгах в базовой валюте символа
SYMBOL_SWAP_MODE_CURRENCY_MARGIN	Свопы начисляются в деньгах в маржинальной валюте символа
SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT	Свопы начисляются в деньгах в валюте депозита клиента
SYMBOL_SWAP_MODE_INTEREST_CURRENT	Свопы начисляются в годовых процентах от цены инструмента на момент расчета свопа(банковский режим - 360 дней в году)
SYMBOL_SWAP_MODE_INTEREST_OPEN	Свопы начисляются в годовых процентах от цены открытия позиции по символу (банковский режим - 360 дней в году)
SYMBOL_SWAP_MODE_REOPEN_CURRENT	Свопы начисляются переоткрытием позиции. В конце торгового дня позиция принудительно закрывается. На следующий день позиция переоткрывается по цене закрытия +/- указанное количество пунктов (в параметрах SYMBOL_SWAP_LONG и SYMBOL_SWAP_SHORT)
SYMBOL_SWAP_MODE_REOPEN_BID	Свопы начисляются переоткрытием позиции. В конце торгового дня позиция принудительно закрывается. На следующий день позиция переоткрывается по текущей цене Bid +/- указанное количество пунктов (в параметрах SYMBOL_SWAP_LONG и SYMBOL_SWAP_SHORT)

Для указания дня недели предназначены значения перечисления ENUM\_DAY\_OF\_WEEK.

#### ENUM\_DAY\_OF\_WEEK

Идентификатор	Описание
---------------	----------

SUNDAY	Воскресенье
MONDAY	Понедельник
TUESDAY	Вторник
WEDNESDAY	Среда
THURSDAY	Четверг
FRIDAY	Пятница
SATURDAY	Суббота

Опцион - это контракт, который дает право, но не обязанность купить или продать базовый актив (товар, акцию, фьючерс и т.д.) по фиксированной цене в течении жизни опциона или в определенный момент времени. Для описания свойств опционов предназначены перечисления, которые описывают тип опциона и право, который он представляет.

#### ENUM\_SYMBOL\_OPTION\_RIGHT

Идентификатор	Описание
SYMBOL_OPTION_RIGHT_CALL	Опцион, дающий право купить актив по фиксированной цене
SYMBOL_OPTION_RIGHT_PUT	Опцион, дающий право продать актив по фиксированной цене

#### ENUM\_SYMBOL\_OPTION\_MODE

Идентификатор	Описание
SYMBOL_OPTION_MODE_EUROPEAN	Европейский тип опциона - может быть погашен только в указанную дату (дата истечения срока, дата исполнения, дата погашения)
SYMBOL_OPTION_MODE_AMERICAN	Американский тип опциона - может быть погашен в любой день до истечения срока опциона. Для такого типа задаётся период, в течение которого покупатель может исполнить данный опцион

## Информация о счете

Для получения информации о текущем счете предназначены функции [AccountInfoInteger\(\)](#), [AccountInfoDouble\(\)](#) и [AccountInfoString\(\)](#). В качестве параметра эти функции принимают значения из соответствующих перечислений ENUM\_ACCOUNT\_INFO.

Для функции [AccountInfoInteger\(\)](#)

### ENUM\_ACCOUNT\_INFO\_INTEGER

Идентификатор	Описание	Тип свойства
ACCOUNT_LOGIN	Номер счета	long
ACCOUNT_TRADE_MODE	Тип торгового счета	<a href="#">ENUM_ACCOUNT_TRADE_MODE</a>
ACCOUNT_LEVERAGE	Размер предоставленного плеча	long
ACCOUNT_LIMIT_ORDERS	Максимально допустимое количество действующих отложенных ордеров	int
ACCOUNT_MARGIN_SO_MODE	Режим задания минимально допустимого уровня залоговых средств	<a href="#">ENUM_ACCOUNT_STOPOUT_MODE</a>
ACCOUNT_TRADE_ALLOWED	<a href="#">Разрешенность торговли</a> для текущего счета	bool
ACCOUNT_TRADE_EXPERT	<a href="#">Разрешенность торговли</a> для эксперта	bool
ACCOUNT_MARGIN_MODE	Режим расчета маржи	<a href="#">ENUM_ACCOUNT_MARGIN_MODE</a>
ACCOUNT_CURRENCY_DIGITS	Количество знаков после запятой для валюты счета, необходимых для точного отображения торговых результатов	int
ACCOUNT_FIFO_CLOSE	Признак того, что позиции можно закрывать только по правилу FIFO. Если значение свойства равно <code>true</code> , то позиции по каждому символу разрешается закрывать только в том порядке, в котором они были открыты — сначала самую старую, затем более новую и т.д. При попытке закрыть позиции в ином порядке будет получена ошибка.	bool

	Для счетов без хеджингового учета позиций ( <a href="#">ACCOUNT_MARGIN_MODE!=ACCOUNT_MARGIN_MODE_RETAIL_HEDGING</a> ) свойство всегда равно false.	
--	--	--

Для функции [AccountInfoDouble\(\)](#)

#### ENUM\_ACCOUNT\_INFO\_DOUBLE

Идентификатор	Описание	Тип свойства
ACCOUNT_BALANCE	Баланс счета в валюте депозита	double
ACCOUNT_CREDIT	Размер предоставленного кредита в валюте депозита	double
ACCOUNT_PROFIT	Размер текущей прибыли на счете в валюте депозита	double
ACCOUNT_EQUITY	Значение собственных средств на счете в валюте депозита	double
ACCOUNT_MARGIN	Размер зарезервированных залоговых средств на счете в валюте депозита	double
ACCOUNT_MARGIN_FREE	Размер свободных средств на счете в валюте депозита, доступных для открытия позиции	double
ACCOUNT_MARGIN_LEVEL	Уровень залоговых средств на счете в процентах	double
ACCOUNT_MARGIN_SO_CALL	Уровень залоговых средств, при котором требуется пополнение счета (Margin Call). В зависимости от установленного ACCOUNT_MARGIN_SO_MODE выражается в процентах либо в валюте депозита	double
ACCOUNT_MARGIN_SO_SO	Уровень залоговых средств, при достижении которого происходит принудительное закрытие самой убыточной позиции (Stop Out). В зависимости от установленного	double

	ACCOUNT_MARGIN_SO_MODE выражается в процентах либо в валюте депозита	
ACCOUNT_MARGIN_INITIAL	Размер средств, зарезервированных на счёте, для обеспечения гарантийной суммы по всем отложенным ордерам	double
ACCOUNT_MARGIN_MAINTENANCE	Размер средств, зарезервированных на счёте, для обеспечения минимальной суммы по всем открытым позициям	double
ACCOUNT_ASSETS	Текущий размер активов на счёте	double
ACCOUNT_LIABILITIES	Текущий размер обязательств на счёте	double
ACCOUNT_COMMISSION_BLOCKED	Текущая сумма заблокированных комиссий по счёту	double

Для функции [AccountInfoString\(\)](#)

#### ENUM\_ACCOUNT\_INFO\_STRING

Идентификатор	Описание	Тип свойства
ACCOUNT_NAME	Имя клиента	string
ACCOUNT_SERVER	Имя торгового сервера	string
ACCOUNT_CURRENCY	Валюта депозита	string
ACCOUNT_COMPANY	Имя компании, обслуживающей счет	string

Существует несколько видов счетов, которые могут быть открыты на торговом сервере. Для того чтобы узнать тип счета, на котором работает MQL5-программа, предназначено перечисление ENUM\_ACCOUNT\_TRADE\_MODE.

#### ENUM\_ACCOUNT\_TRADE\_MODE

Идентификатор	Описание
ACCOUNT_TRADE_MODE_DEMO	Демонстрационный торговый счет
ACCOUNT_TRADE_MODE_CONTEST	Конкурсный торговый счет
ACCOUNT_TRADE_MODE_REAL	Реальный торговый счет

При нехватке собственных средств для поддержания открытых позиций возникает ситуация принудительного закрытия Stop Out. Минимальный уровень маржи, при котором наступает Stop Out, может задаваться в процентах или в денежном выражении. Узнать какой режим задан для данного счета можно с помощью перечисления ENUM\_ACCOUNT\_STOPOUT\_MODE.

#### ENUM\_ACCOUNT\_STOPOUT\_MODE

Идентификатор	Описание
ACCOUNT_STOPOUT_MODE_PERCENT	Уровень задается в процентах
ACCOUNT_STOPOUT_MODE_MONEY	Уровень задается в деньгах

#### ENUM\_ACCOUNT\_MARGIN\_MODE

Идентификатор	Описание
ACCOUNT_MARGIN_MODE_RETAIL_NETTING	Используется для внебиржевого рынка при учете позиций в режиме "неттинг" (по одному символу может быть только одна позиция). Расчет маржи осуществляется на основе типа инструмента ( <a href="#">SYMBOL_TRADE_CALC_MODE</a> ).
ACCOUNT_MARGIN_MODE_EXCHANGE	Используется для биржевого рынка. Расчет маржи осуществляется на основе дисконтов, указанных в настройках инструментов. Дисконты устанавливаются брокером, однако не могут быть ниже значений, определенных биржей.
ACCOUNT_MARGIN_MODE_RETAIL_HEDGING	Используется для внебиржевого рынка при независимом учете позиций ("хеджинг", по одному символу может быть несколько позиций). Расчет маржи осуществляется на основе типа инструмента ( <a href="#">SYMBOL_TRADE_CALC_MODE</a> ) и с учетом размера захеджированной маржи ( <a href="#">SYMBOL_MARGIN_HEDGED</a> ).

Пример скрипта, выводящего краткую информацию о счете.

```
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- имя компании
string company=AccountInfoString(ACCOUNT_COMPANY);
//--- имя клиента
string name=AccountInfoString(ACCOUNT_NAME);
//--- номер счета
long login=AccountInfoInteger(ACCOUNT_LOGIN);
//--- имя сервера
```

```
string server=AccountInfoString(ACCOUNT_SERVER);
//--- валюта счета
string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- демо, конкурсный или реальный счет
ENUM_ACCOUNT_TRADE_MODE account_type=(ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOUNT_TYPE);
//--- теперь превратим значение перечисления в понятный вид
string trade_mode;
switch(account_type)
{
    case ACCOUNT_TRADE_MODE_DEMO:
        trade_mode="demo";
        break;
    case ACCOUNT_TRADE_MODE_CONTEST:
        trade_mode="конкурсный";
        break;
    default:
        trade_mode="реальный";
        break;
}
//--- в процентах или в денежном выражении задается уровень Stop Out
ENUM_ACCOUNT_STOPOUT_MODE stop_out_mode=(ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_STOPOUT_MODE);
//--- получим значения уровней, при которых наступает Margin Call и Stop Out
double margin_call=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL);
double stop_out=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO);
//--- выведем краткую информацию по счету
PrintFormat("Счет клиента '%s' # %d %s открыт в '%s' на сервере '%s'",
            name,login,trade_mode,company,server);
PrintFormat("Валюта счета - %s, уровень MarginCall и StopOut задается в %s",
            currency,(stop_out_mode==ACCOUNT_STOPOUT_MODE_PERCENT)?"процентах":" в ");
PrintFormat("Уровень MarginCall=%G, уровень StopOut=%G",margin_call,stop_out);
}
```

## Статистика тестирования

После окончания тестирования вычисляются статистические показатели результатов торговли по множеству параметров. Значения показателей можно получить с помощью функции [TesterStatistics\(\)](#), указав идентификатор показателя из перечисления ENUM\_STATISTICS.

Хотя при вычислении статистики используются показатели двух типов - int и double - функция возвращает все значения в виде double. Все статистические величины, имеющие тип double, по умолчанию выражаются в валюте депозита, если не сказано иное.

### ENUM\_STATISTICS

Идентификатор	Описание статистического показателя	Тип
STAT_INITIAL_DEPOSIT	Значение начального депозита	double
STAT_WITHDRAWAL	Количество выведенных со счета средств	double
STAT_PROFIT	Чистая прибыль по окончании тестирования, сумма STAT_GROSS_PROFIT и STAT_GROSS_LOSS (STAT_GROSS_LOSS всегда меньше или равно нулю)	double
STAT_GROSS_PROFIT	Общая прибыль, сумма всех прибыльных (положительных) трейдов. Значение больше или равно нулю	double
STAT_GROSS_LOSS	Общий убыток, сумма всех убыточных (отрицательных) трейдов. Значение меньше или равно нулю	double
STAT_MAX_PROFITTRADE	Максимальная прибыль - наибольшее значение среди всех прибыльных трейдов. Значение больше или равно нулю	double
STAT_MAX_LOSSTRADE	Максимальный убыток - наименьшее значение среди всех убыточных трейдов. Значение меньше или равно нулю	double
STAT_CONPROFITMAX	Максимальная прибыль в последовательности прибыльных трейдов.	double

	Значение больше или равно нулю	
STAT_CONPROFITMAX_TRADES	Количество трейдов, сформировавших STAT_CONPROFITMAX (максимальная прибыль в последовательности прибыльных трейдов)	int
STAT_MAX_CONWINS	Общая прибыль в самой длинной серии прибыльных трейдов	double
STAT_MAX_CONPROFIT_TRADES	Количество трейдов в самой длинной серии прибыльных трейдов STAT_MAX_CONWINS	int
STAT_CONLOSSMAX	Максимальный убыток в последовательности убыточных трейдов. Значение меньше или равно нулю	double
STAT_CONLOSSMAX_TRADES	Количество трейдов, сформировавших STAT_CONLOSSMAX (максимальный убыток в последовательности убыточных трейдов)	int
STAT_MAX_CONLOSSES	Общий убыток в самой длинной серии убыточных трейдов	double
STAT_MAX_CONLOSS_TRADES	Количество трейдов в самой длинной серии убыточных трейдов STAT_MAX_CONLOSSES	int
STAT_BALANCEMIN	Минимальное значение баланса	double
STAT_BALANCE_DD	Максимальная просадка баланса в деньгах. В процессе торговли баланс может испытать множество просадок, берется наибольшее значение.	double
STAT_BALANCEDD_PERCENT	Просадка баланса в процентах, которая была зафиксирована в момент максимальной просадки	double

	баланса в деньгах (STAT_BALANCE_DD).	
STAT_BALANCE_DDREL_PERCENT	Максимальная просадка баланса в процентах. В процессе торговли баланс может испытать множество просадок, для каждой фиксируется относительное значение просадки в процентах. Возвращается наибольшее значение	double
STAT_BALANCE_DD_RELATIVE	Просадка баланса в деньгах, которая была зафиксирована в момент максимальной просадки баланса в процентах (STAT_BALANCE_DDREL_PERCENT).	double
STAT_EQUITYMIN	Минимальное значение собственных средств	double
STAT_EQUITY_DD	Максимальная просадка средств в деньгах. В процессе торговли средства могут испытать множество просадок, берется наибольшее значение.	double
STAT_EQUITYDD_PERCENT	Просадка средств в процентах, которая была зафиксирована в момент максимальной просадки средств в деньгах (STAT_EQUITY_DD).	double
STAT_EQUITY_DDREL_PERCENT	Максимальная просадка средств в процентах. В процессе торговли средства могут испытать множество просадок, для каждой фиксируется относительное значение просадки в процентах. Возвращается наибольшее значение	double
STAT_EQUITY_DD_RELATIVE	Просадка средств в деньгах, которая была зафиксирована в момент максимальной просадки средств в процентах	double

	(STAT_EQUITY_DDREL_PERCENT).	
STAT_EXPECTED_PAYOFF	Математическое ожидание выигрыша	double
STAT_PROFIT_FACTOR	Прибыльность - отношение STAT_GROSS_PROFIT/STAT_GROSS_LOSS. Если STAT_GROSS_LOSS=0, то прибыльность принимает значение <a href="#">DBL_MAX</a>	double
STAT_RECOVERY_FACTOR	Фактор восстановления - отношение STAT_PROFIT/STAT_BALANCE_DD	double
STAT_SHARPE_RATIO	Коэффициент Шарпа	double
STAT_MIN_MARGINLEVEL	Минимальное достигнутое значение уровня маржи	double
STAT_CUSTOM_ONTESTER	Значение рассчитанного пользовательского критерия оптимизации, возвращаемого функцией <a href="#">OnTester()</a>	double
STAT DEALS	Количество совершенных сделок	int
STAT TRADES	Количество трейдов	int
STAT PROFIT TRADES	Прибыльные трейды	int
STAT LOSS TRADES	Убыточные трейды	int
STAT SHORT TRADES	Короткие трейды	int
STAT LONG TRADES	Длинные трейды	int
STAT PROFIT_SHORTTRADES	Короткие прибыльные трейды	int
STAT PROFIT_LONGTRADES	Длинные прибыльные трейды	int
STAT PROFITTRADES_AVGCON	Средняя длина прибыльной серии трейдов	int
STAT LOSSTRADES_AVGCON	Средняя длина убыточной серии трейдов	int

## Торговые константы

Разнообразные константы, используемые для программирования торговых стратегий, разделены на следующие группы:

- [Информация об исторических данных по инструменту](#) - получение общей информации по финансовому инструменту;
- [Свойства ордеров](#) - получение информации о торговых ордерах;
- [Свойства позиций](#) - получение информации о текущих позициях;
- [Свойства сделок](#) - получение информации о совершенных сделках;
- [Типы торговых операций](#) - описание доступных торговых операций;
- [Типы торговых транзакций](#) - описание возможных типов торговых транзакций;
- [Виды заявок в стакане цен](#) - для разделения заявок по направлению запрашиваемой торговой операции.

## Информация об исторических данных по инструменту

При [доступе к таймсериям](#) для получения дополнительной [информации об инструменте](#) используется функция [SeriesInfoInteger\(\)](#). В качестве параметра этой функции передается идентификатор требуемого свойства, который может принимать одно из значений перечисления ENUM\_SERIES\_INFO\_INTEGER.

### ENUM\_SERIES\_INFO\_INTEGER

Идентификатор	Описание	Тип свойства
SERIES_BARS_COUNT	Количество баров по символу-периоду на данный момент	long
SERIES_FIRSTDATE	Самая первая дата по символу-периоду на данный момент	datetime
SERIES_LASTBAR_DATE	Время открытия последнего бара по символу-периоду	datetime
SERIES_SERVER_FIRSTDATE	Самая первая дата в истории по символу на сервере независимо от периода	datetime
SERIES_TERMINAL_FIRSTDATE	Самая первая дата в истории по символу в клиентском терминале независимо от периода	datetime
SERIES_SYNCHRONIZED	Признак синхронизированности данных по символу/периоду на данный момент	bool

## Свойства ордеров

Приказы на проведение торговых операций оформляются ордерами. Каждый ордер имеет множество свойств для чтения, информацию по ним можно получать с помощью функций [OrderGet...\(\)](#) и [HistoryOrderGet...\(\)](#).

Для функций [OrderGetInteger\(\)](#) и [HistoryOrderGetInteger\(\)](#)

### ENUM\_ORDER\_PROPERTY\_INTEGER

Идентификатор	Описание	Тип
ORDER_TICKET	Тикет ордера. Уникальное число, которое присваивается каждому ордеру	long
ORDER_TIME_SETUP	Время постановки ордера	datetime
ORDER_TYPE	Тип ордера	<a href="#">ENUM_ORDER_TYPE</a>
ORDER_STATE	Статус ордера	<a href="#">ENUM_ORDER_STATE</a>
ORDER_TIME_EXPIRATION	Время истечения ордера	datetime
ORDER_TIME_DONE	Время исполнения или снятия ордера	datetime
ORDER_TIME_SETUP_MSC	Время установки ордера на исполнение в миллисекундах с 01.01.1970	long
ORDER_TIME_DONE_MSC	Время исполнения/снятия ордера в миллисекундах с 01.01.1970	long
ORDER_TYPE_FILLING	Тип исполнения по остатку	<a href="#">ENUM_ORDER_TYPE_FILLING</a>
ORDER_TYPE_TIME	Время жизни ордера	<a href="#">ENUM_ORDER_TYPE_TIME</a>
ORDER_MAGIC	Идентификатор эксперта выставившего ордер (предназначен для того, чтобы каждый эксперт выставлял свой собственный уникальный номер)	long
ORDER_REASON	Причина или источник выставления ордера	<a href="#">ENUM_ORDER_REASON</a>
ORDER_POSITION_ID	Идентификатор позиции, который ставится на ордере при его исполнении. Каждый выполненный ордер порождает <a href="#"> сделку</a> , которая открывает новую или	long

	изменяет уже существующую <a href="#">позицию</a> . Идентификатор этой позиции и устанавливается исполненному ордеру в этот момент.	
ORDER_POSITION_BY_ID	Идентификатор встречной позиции для ордеров типа ORDER_TYPE_CLOSE_BY.	long

Для функций [OrderGetDouble\(\)](#) и [HistoryOrderGetDouble\(\)](#)

#### ENUM\_ORDER\_PROPERTY\_DOUBLE

Идентификатор	Описание	Тип
ORDER_VOLUME_INITIAL	Первоначальный объем при постановке ордера	double
ORDER_VOLUME_CURRENT	Невыполненный объем	double
ORDER_PRICE_OPEN	Цена, указанная в ордере	double
ORDER_SL	Уровень Stop Loss	double
ORDER_TP	Уровень Take Profit	double
ORDER_PRICE_CURRENT	Текущая цена по символу ордера	double
ORDER_PRICE_STOPLIMIT	Цена постановки Limit ордера при срабатывании StopLimit ордера	double

Для функций [OrderGetString\(\)](#) и [HistoryOrderGetString\(\)](#)

#### ENUM\_ORDER\_PROPERTY\_STRING

Идентификатор	Описание	Тип
ORDER_SYMBOL	Символ, по которому выставлен ордер	string
ORDER_COMMENT	Комментарий	string
ORDER_EXTERNAL_ID	Идентификатор ордера во внешней торговой системе (на бирже)	string

При отправке торгового запроса функцией [OrderSend\(\)](#) для некоторых операций необходимо указать тип ордера. Тип ордера указывается в поле *type* специальной структуры [MqlTradeRequest](#), и может принимать значения из перечисления ENUM\_ORDER\_TYPE.

## ENUM\_ORDER\_TYPE

Идентификатор	Описание
ORDER_TYPE_BUY	Рыночный ордер на покупку
ORDER_TYPE_SELL	Рыночный ордер на продажу
ORDER_TYPE_BUY_LIMIT	Отложенный ордер Buy Limit
ORDER_TYPE_SELL_LIMIT	Отложенный ордер Sell Limit
ORDER_TYPE_BUY_STOP	Отложенный ордер Buy Stop
ORDER_TYPE_SELL_STOP	Отложенный ордер Sell Stop
ORDER_TYPE_BUY_STOP_LIMIT	По достижении цены ордера выставляется отложенный ордер Buy Limit по цене StopLimit
ORDER_TYPE_SELL_STOP_LIMIT	По достижении цены ордера выставляется отложенный ордер Sell Limit по цене StopLimit
ORDER_TYPE_CLOSE_BY	Ордер на закрытие позиции встречной позицией

Каждый ордер имеет статус, описывающий его состояние. Для получения информации используйте функцию [OrderGetInteger\(\)](#) или [HistoryOrderGetInteger\(\)](#) с модификатором ORDER\_STATE. Допустимые значения хранятся в перечислении ENUM\_ORDER\_STATE.

## ENUM\_ORDER\_STATE

Идентификатор	Описание
ORDER_STATE_STARTED	Ордер проверен на корректность, но еще не принят брокером
ORDER_STATE_PLACED	Ордер принят
ORDER_STATE_CANCELED	Ордер снят клиентом
ORDER_STATE_PARTIAL	Ордер выполнен частично
ORDER_STATE_FILLED	Ордер выполнен полностью
ORDER_STATE_REJECTED	Ордер отклонен
ORDER_STATE_EXPIRED	Ордер снят по истечении срока его действия
ORDER_STATE_REQUEST_ADD	Ордер в состоянии регистрации (выставление в торговую систему)
ORDER_STATE_REQUEST MODIFY	Ордер в состоянии модификации (изменение его параметров)

ORDER_STATE_REQUEST_CANCEL	Ордер в состоянии удаления (удаление из торговой системы)
----------------------------	---

При отправке торгового запроса функцией [OrderSend\(\)](#) для ордера можно задать политику исполнения в поле *type\_filling* в специальной структуре [MqlTradeRequest](#), допустимы значения из перечисления [ENUM\\_ORDER\\_TYPE\\_FILLING](#). Для получения значения этого свойства используйте функцию [OrderGetInteger\(\)](#) или [HistoryOrderGetInteger\(\)](#) с модификатором [ORDER\\_TYPE\\_FILLING](#).

#### ENUM\_ORDER\_TYPE\_FILLING

Идентификатор	Описание
ORDER_FILLING_FOK	Данная политика исполнения означает, что ордер может быть выполнен исключительно в указанном объеме. Если на рынке в данный момент не присутствует достаточного объема финансового инструмента, то ордер не будет выполнен. Необходимый объем может быть составлен из нескольких предложений, доступных в данный момент на рынке.
ORDER_FILLING_IOC	Означает согласие совершить сделку по максимально доступному на рынке объему в пределах указанного в ордере. В случае невозможности полного исполнения ордер будет выполнен на доступный объем, а неисполненный объем ордера будет отменен.
ORDER_FILLING_RETURN	Данный режим используется для рыночных (ORDER_TYPE_BUY и ORDER_TYPE_SELL), лимитных и стоп-лимитных ордеров (ORDER_TYPE_BUY_LIMIT, ORDER_TYPE_SELL_LIMIT, ORDER_TYPE_BUY_STOP_LIMIT и ORDER_TYPE_SELL_STOP_LIMIT) и только в <a href="#">режимах</a> "Исполнение по рынку" и "Биржевое исполнение". В случае частичного исполнения рыночный или лимитный ордер с остаточным объемом не снимается, а продолжает действовать. Для ордеров ORDER_TYPE_BUY_STOP_LIMIT и ORDER_TYPE_SELL_STOP_LIMIT при активации будет создан соответствующий лимитный ордер ORDER_TYPE_BUY_LIMIT/ORDER_TYPE_SELL_LIMIT с типом исполнения ORDER_FILLING_RETURN.

Срок действия ордера можно задать в поле `type_time` специальной структуры [MqlTradeRequest](#) при отправке торгового запроса функцией [OrderSend\(\)](#). Допустимы значения из перечисления `ENUM_ORDER_TYPE_TIME`. Для получения значения этого свойства используйте функцию [OrderGetInteger\(\)](#) или [HistoryOrderGetInteger\(\)](#) с модификатором `ORDER_TYPE_TIME`.

#### ENUM\_ORDER\_TYPE\_TIME

Идентификатор	Описание
<code>ORDER_TIME_GTC</code>	Ордер будет находиться в очереди до тех пор, пока не будет снят
<code>ORDER_TIME_DAY</code>	Ордер будет действовать только в течение текущего торгового дня
<code>ORDER_TIME_SPECIFIED</code>	Ордер будет действовать до даты истечения
<code>ORDER_TIME_SPECIFIED_DAY</code>	Ордер будет действовать до 23:59:59 указанного дня. Если это время не попадает на торговую сессию, истечение наступит в ближайшее торговое время.

В свойстве `ORDER_REASON` содержится причина выставления ордера. Ордер может быть выставлен с помощью MQL5 программы, или из мобильного приложения, или в результате наступления события `StopOut`, и т.д. Возможные значения `ORDER_REASON` описываются в перечислении `ENUM_ORDER_REASON`.

#### ENUM\_ORDER\_REASON

Идентификатор	Описание
<code>ORDER_REASON_CLIENT</code>	Ордер выставлен из десктопного терминала
<code>ORDER_REASON_MOBILE</code>	Ордер выставлен из мобильного приложения
<code>ORDER_REASON_WEB</code>	Ордер выставлен из веб-платформы
<code>ORDER_REASON_EXPERT</code>	Ордер выставлен из MQL5-программы - советником или скриптом
<code>ORDER_REASON_SL</code>	Ордер выставлен в результате срабатывания Stop Loss
<code>ORDER_REASON_TP</code>	Ордер выставлен в результате срабатывания Take Profit
<code>ORDER_REASON_SO</code>	Ордер выставлен в результате наступления события <code>Stop Out</code>

## Свойства позиций

Результатом совершения [торговых операций](#) являются открытие позиции, изменение её объема и/или направления, или ее ликвидация. Торговые операции проводятся на основание [ордеров](#), отправляемых функцией [OrderSend\(\)](#) в виде [торговых запросов](#). Для каждого финансового [инструмента](#) (символа) возможна только одна открытая позиция. Позиция имеет набор свойств, доступных для чтений функциями [PositionGet...\(\)](#).

Для функции [PositionGetInteger\(\)](#)

### ENUM\_POSITION\_PROPERTY\_INTEGER

Идентификатор	Описание	Тип
POSITION_TICKET	<p>Тикет позиции. Уникальное число, которое присваивается каждой вновь открытой позиции. Как правило, соответствует тикету ордера, в результате которого она была открыта, за исключением случаев изменения тикета в результате служебных операций на сервере. Например, начисления свопов переоткрытием позиции. Для нахождения ордера, которым была открыта позиция, следует использовать свойство POSITION_IDENTIFIER.</p> <p>Значение POSITION_TICKET соответствует <a href="#">MqlTradeRequest::position</a>.</p>	long
POSITION_TIME	Время открытия позиции	datetime
POSITION_TIME_MSC	Время открытия позиции в миллисекундах с 01.01.1970	long
POSITION_TIME_UPDATE	Время изменения позиции в секундах с 01.01.1970	long
POSITION_TIME_UPDATE_MSC	Время изменения позиции в миллисекундах с 01.01.1970	long
POSITION_TYPE	Тип позиции	<a href="#">ENUM_POSITION_TYPE</a>
POSITION_MAGIC	Magic number для позиции (смотри <a href="#">ORDER_MAGIC</a> )	long

POSITION_IDENTIFIER	<p>Идентификатор позиции - это уникальное число, которое присваивается каждой вновь открытой позиции и не изменяется в течение всей ее жизни. Соответствует тикету ордера, которым была открыта позиция.</p> <p>Идентификатор позиции указывается в каждом ордере (ORDER_POSITION_ID) и сделке (DEAL_POSITION_ID), которая ее открыла, изменила или закрыла. Используйте это свойство для поиска ордеров и сделок, связанных с позицией.</p> <p>При развороте позиции в режиме неттинга (единой сделкой in/out) идентификатор позиции POSITION_IDENTIFIER не изменяется. Однако при этом POSITION_TICKET изменяется на тикет ордера, в результате которого произошел разворот. В режиме хеджинга разворот позиции не предусмотрен.</p>	long
POSITION_REASON	Причина открытия позиции	<a href="#">ENUM_POSITION_REASON</a>

Для функции [PositionGetDouble\(\)](#)

#### ENUM\_POSITION\_PROPERTY\_DOUBLE

Идентификатор	Описание	Тип
POSITION_VOLUME	Объем позиции	double
POSITION_PRICE_OPEN	Цена позиции	double
POSITION_SL	Уровень Stop Loss для открытой позиции	double
POSITION_TP	Уровень Take Profit для открытой позиции	double
POSITION_PRICE_CURRENT	Текущая цена по символу	double
POSITION_SWAP	Накопленный своп	double

POSITION_PROFIT	Текущая прибыль	double
-----------------	-----------------	--------

Для функции [PositionGetString\(\)](#)

#### ENUM\_POSITION\_PROPERTY\_STRING

Идентификатор	Описание	Тип
POSITION_SYMBOL	Символ, по которому открыта позиция	string
POSITION_COMMENT	Комментарий к позиции	string
POSITION_EXTERNAL_ID	Идентификатор позиции во внешней системе (на бирже)	string

Направление открытой позиции (покупка или продажа) определяется значением из перечисления ENUM\_POSITION\_TYPE. Для получения типа открытой позиции используйте функцию [PositionGetInteger\(\)](#) с модификатором POSITION\_TYPE.

#### ENUM\_POSITION\_TYPE

Идентификатор	Описание
POSITION_TYPE_BUY	Покупка
POSITION_TYPE_SELL	Продажа

В свойстве POSITION\_REASON содержится причина открытия позиции. Позиция может быть открыта в результате срабатывания ордера, который был выставлен из десктопного терминала, из мобильного приложения, с помощью советника и т.д. Возможные значения POSITION\_REASON описываются в перечислении ENUM\_POSITION\_REASON.

#### ENUM\_POSITION\_REASON

Идентификатор	Описание
POSITION_REASON_CLIENT	Позиция открыта в результате срабатывания ордера, выставленного из десктопного терминала
POSITION_REASON_MOBILE	Позиция открыта в результате срабатывания ордера, выставленного из мобильного приложения
POSITION_REASON_WEB	Позиция открыта в результате срабатывания ордера, выставленного из веб-платформы
POSITION_REASON_EXPERT	Позиция открыта в результате срабатывания ордера, выставленного из MQL5-программы - советником или скриптом

## Свойства сделок

Сделка является отражением факта совершения [торговой операции](#) на основании [ордера](#), содержащего торговый приказ. Каждая сделка описывается свойствами, позволяющими получить информацию о ней. Для чтения значений свойств используются функции вида [HistoryDealGet...\(\)](#), возвращающие значения из соответствующих перечислений.

Для функции [HistoryDealGetInteger\(\)](#)

### ENUM DEAL PROPERTY INTEGER

Идентификатор	Описание	Тип
DEAL_TICKET	Тикет сделки. Уникальное число, которое присваивается каждой сделке	long
DEAL_ORDER	<a href="#">Ордер</a> , на основание которого выполнена сделка	long
DEAL_TIME	Время совершения сделки	datetime
DEAL_TIME_MSC	Время совершения сделки в миллисекундах с 01.01.1970	long
DEAL_TYPE	Тип сделки	<a href="#">ENUM DEAL TYPE</a>
DEAL_ENTRY	Направление сделки - вход в рынок, выход из рынка или разворот	<a href="#">ENUM DEAL ENTRY</a>
DEAL_MAGIC	Magic number для сделки (смотри <a href="#">ORDER MAGIC</a> )	long
DEAL_REASON	Причина или источник проведения сделки	<a href="#">ENUM DEAL REASON</a>
DEAL_POSITION_ID	<a href="#">Идентификатор позиции</a> , в открытии, изменении или закрытии которой участвовала эта сделка. Каждая позиция имеет уникальный идентификатор, который присваивается всем сделкам, совершенным на инструменте в течение всей жизни позиции.	long

Для функции [HistoryDealGetDouble\(\)](#)

### ENUM DEAL PROPERTY DOUBLE

Идентификатор	Описание	Тип
---------------	----------	-----

DEAL_VOLUME	Объем сделки	double
DEAL_PRICE	Цена сделки	double
DEAL_COMMISSION	Комиссия по сделке	double
DEAL_SWAP	Накопленный своп при закрытии	double
DEAL_PROFIT	Финансовый результат сделки	double

Для функции [HistoryDealGetString\(\)](#)

#### ENUM DEAL PROPERTY STRING

Идентификатор	Описание	Тип
DEAL_SYMBOL	Имя символа, по которому произведена сделка	string
DEAL_COMMENT	Комментарий к сделке	string
DEAL_EXTERNAL_ID	Идентификатор сделки во внешней торговой системе (на бирже)	string

Каждая сделка характеризуется типом, возможные значения перечислены в ENUM DEAL TYPE. Для получения информации о типе сделки используйте функцию [HistoryDealGetInteger\(\)](#) с модификатором DEAL\_TYPE.

#### ENUM DEAL TYPE

Идентификатор	Описание
DEAL_TYPE_BUY	Покупка
DEAL_TYPE_SELL	Продажа
DEAL_TYPE_BALANCE	Начисление баланса
DEAL_TYPE_CREDIT	Начисление кредита
DEAL_TYPE_CHARGE	Дополнительные сборы
DEAL_TYPE_CORRECTION	Корректирующая запись
DEAL_TYPE_BONUS	Перечисление бонусов
DEAL_TYPE_COMMISSION	Дополнительные комиссии
DEAL_TYPE_COMMISSION_DAILY	Комиссия, начисляемая в конце торгового дня
DEAL_TYPE_COMMISSION_MONTHLY	Комиссия, начисляемая в конце месяца

DEAL_TYPE_COMMISSION_AGENT_DAILY	Агентская комиссия, начисляемая в конце торгового дня
DEAL_TYPE_COMMISSION_AGENT_MONTHLY	Агентская комиссия, начисляемая в конце месяца
DEAL_TYPE_INTEREST	Начисления процентов на свободные средства
DEAL_TYPE_BUY_CANCELED	Отмененная сделка покупки. Возможная ситуация, когда ранее совершенная сделка на покупку отменяется. В таком случае тип ранее совершенной сделки (DEAL_TYPE_BUY) меняется на DEAL_TYPE_BUY_CANCELED, а ее прибыль/убыток обнуляется. Ранее полученная прибыль/убыток начисляется/ списывается со счета отдельной балансовой операцией
DEAL_TYPE_SELL_CANCELED	Отмененная сделка продажи. Возможная ситуация, когда ранее совершенная сделка на продажу отменяется. В таком случае тип ранее совершенной сделки (DEAL_TYPE_SELL) меняется на DEAL_TYPE_SELL_CANCELED, а ее прибыль/убыток обнуляется. Ранее полученная прибыль/убыток начисляется/ списывается со счета отдельной балансовой операцией
DEAL_DIVIDEND	Начисление дивиденда
DEAL_DIVIDEND_FRANKED	Начисление франкированного дивиденда (освобожденного от уплаты налога)
DEAL_TAX	Начисление налога

Сделки различаются не только по типу, задаваемого в перечислении ENUM DEAL\_TYPE, но и по способу изменения позиции. Это может быть простое открытие позиции или наращивание объема ранее открытой позиции ( вход в рынок), закрытие позиции сделкой противоположного направления соответствующим объемом (выход из рынка) или переворот позиции в том случае, когда объем сделки в противоположном направлении перекрывает объем ранее открытой позиции.

Все эти ситуации описаны значениями из перечисления ENUM DEAL\_ENTRY. Для получения этой информации о сделке используйте функцию [HistoryDealGetInteger\(\)](#) с модификатором DEAL\_ENTRY.

#### ENUM DEAL\_ENTRY

Идентификатор	Описание
DEAL_ENTRY_IN	Вход в рынок
DEAL_ENTRY_OUT	Выход из рынка

DEAL_ENTRY_INOUT	Разворот
DEAL_ENTRY_OUT_BY	Закрытие встречной позицией

В свойстве DEAL\_REASON содержится причина проведения сделки. Сделка может быть проведена в результате срабатывания ордера, выставленного из мобильного приложения или из MQL5 программы; либо в результате наступления события StopOut или начисления/ списания вариационной маржи, и т.д. Возможные значения DEAL\_REASON описываются в перечислении ENUM DEAL\_REASON. Для неторговых сделок, вызванных операциями изменения баланса, кредита, начисления комиссий и прочих, в качестве причины указывается DEAL\_REASON\_CLIENT.

#### ENUM DEAL\_REASON

Идентификатор	Описание
DEAL_REASON_CLIENT	Сделка проведена в результате срабатывания ордера, выставленного из десктопного терминала
DEAL_REASON_MOBILE	Сделка проведена в результате срабатывания ордера, выставленного из мобильного приложения
DEAL_REASON_WEB	Сделка проведена в результате срабатывания ордера, выставленного из веб-платформы
DEAL_REASON_EXPERT	Сделка проведена в результате срабатывания ордера, выставленного из MQL5-программы - советником или скриптом
DEAL_REASON_SL	Сделка проведена в результате срабатывания ордера Stop Loss
DEAL_REASON_TP	Сделка проведена в результате срабатывания ордера Take Profit
DEAL_REASON_SO	Сделка проведена в результате наступления события Stop Out
DEAL_REASON_ROLLOVER	Сделка проведена по причине переноса позиции
DEAL_REASON_VMargin	Сделка проведена по причине начисления/ списания вариационной маржи
DEAL_REASON_SPLIT	Сделка проведена по причине сплита (понижения цены) инструмента, по которому имелась позиция на момент проведения сплита

## Типы торговых операций

Торговля осуществляется посредством отправки с помощью функции `OrderSend()` приказов на открытие позиций, а также приказов на установку, модификацию и удаление отложенных ордеров. Каждый торговый приказ содержит указание на тип запрашиваемой торговой операции. Торговые операции описаны в перечислении `ENUM_TRADE_REQUEST_ACTIONS`.

### `ENUM_TRADE_REQUEST_ACTIONS`

Идентификатор	Описание
<code>TRADE_ACTION DEAL</code>	Установить торговый ордер на немедленное совершение сделки с указанными параметрами (поставить рыночный ордер)
<code>TRADE_ACTION PENDING</code>	Установить торговый ордер на совершение сделки при указанных условиях (отложенный ордер)
<code>TRADE_ACTION SLTP</code>	Изменить значения Stop Loss и Take Profit у открытой позиции
<code>TRADE_ACTION MODIFY</code>	Изменить параметры ранее установленного торгового ордера
<code>TRADE_ACTION REMOVE</code>	Удалить ранее выставленный отложенный торговый ордер
<code>TRADE_ACTION CLOSE_BY</code>	Закрыть позицию встречной

Пример торговой операции `TRADE_ACTION DEAL` для открытия позиции Buy:

```
#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Открытие позиции Buy
//+-----+
void OnStart()
{
//--- объявление и инициализация запроса и результата
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
//--- параметры запроса
    request.action    =TRADE_ACTION_DEAL;           // тип торговой операции
    request.symbol    =Symbol();                    // символ
    request.volume    =0.1;                         // объем в 0.1 лот
    request.type      =ORDER_TYPE_BUY;             // тип ордера
    request.price     =SymbolInfoDouble(Symbol(),SYMBOL_ASK); // цена для открытия
    request.deviation=5;                          // допустимое отклонение
    request.magic     =EXPERT_MAGIC;               // MagicNumber ордера
//--- отправка запроса
    if(!OrderSend(request,result))                // если отправить запрос
        PrintFormat("OrderSend error %d",GetLastError()); // вывести ошибку
//--- информация об операции
    PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
//+-----+
```

Пример торговой операции **TRADE\_ACTION DEAL** для открытия позиции Sell:

```
#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Открытие позиции Sell
//+-----+
void OnStart()
{
    //--- объявление и инициализация запроса и результата
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
    //--- параметры запроса
    request.action =TRADE_ACTION_DEAL; // тип торговой операции
    request.symbol =Symbol(); // символ
    request.volume =0.2; // объем в 0.2 лот
    request.type =ORDER_TYPE_SELL; // тип ордера
    request.price =SymbolInfoDouble(Symbol(),SYMBOL_BID); // цена для открытия
    request.deviation=5; // допустимое отклонение
    request.magic =EXPERT_MAGIC; // MagicNumber ордера
    //--- отправка запроса
    if(!OrderSend(request,result))
        PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
    //--- информация об операции
    PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
//+-----+
```

Пример торговой операции **TRADE\_ACTION DEAL** для закрытия позиций:

```

#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Закрытие всех позиций |
//+-----+
void OnStart()
{
//--- объявление запроса и результата
    MqlTradeRequest request;
    MqlTradeResult result;
    int total=PositionsTotal(); // количество открытых позиций
//--- перебор всех открытых позиций
    for(int i=total-1; i>=0; i--)
    {
//--- параметры ордера
        ulong position_ticket=PositionGetTicket(i);
        string position_symbol=PositionGetString(POSITION_SYMBOL);
        int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
        ulong magic=PositionGetInteger(POSITION_MAGIC);
        double volume=PositionGetDouble(POSITION_VOLUME);
        ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
//--- вывод информации о позиции
        PrintFormat("#%I64u %s %.2f %s [%I64d]",
                    position_ticket,
                    position_symbol,
                    EnumToString(type),
                    volume,
                    DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                    magic);
//--- если MagicNumber совпадает
        if(magic==EXPERT_MAGIC)
        {
//--- обнуление значений запроса и результата
            ZeroMemory(request);
            ZeroMemory(result);
//--- установка параметров операции
            request.action =TRADE_ACTION_DEAL; // тип торговой операции
            request.position =position_ticket; // тикет позиции
            request.symbol =position_symbol; // символ
            request.volume =volume; // объем позиции
            request.deviation=5; // допустимое отклонение от цены
            request.magic =EXPERT_MAGIC; // MagicNumber позиции
//--- установка цены и типа ордера в зависимости от типа позиции
            if(type==POSITION_TYPE_BUY)
            {
                request.price=SymbolInfoDouble(position_symbol,SYMBOL_BID);
                request.type =ORDER_TYPE_SELL;
            }
            else
            {
                request.price=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
                request.type =ORDER_TYPE_BUY;
            }
//--- вывод информации о закрытии
            PrintFormat("Close %I64d %s %s",position_ticket,position_symbol,EnumToString(type));
//--- отправка запроса
            if(!OrderSend(request,result))
                PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
//--- информация об операции
            PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,
//---
        }
    }
}

```

```

    }
}

```

Пример торговой операции **TRADE\_ACTION\_PENDING** для установки отложенного ордера:

```

#property description "Пример установки отложенных ордеров"
#property script_show_inputs
#define EXPERT_MAGIC 123456                                     // MagicNumber эксперта
input ENUM_ORDER_TYPE orderType=ORDER_TYPE_BUY_LIMIT;        // тип ордера
//+-----+
//| Установка отложенных ордеров                           |
//+-----+
void OnStart()
{
//-- объявление и инициализация запроса и результата
MqlTradeRequest request={0};
MqlTradeResult result={0};
//-- параметры для установки отложенного ордера
request.action =TRADE_ACTION_PENDING;                         // тип торговой
request.symbol =Symbol();                                    // символ
request.volume =0.1;                                       // объем в 0.1
request.deviation=2;                                       // допустимое с
request.magic =EXPERT_MAGIC;                                // MagicNumber
int offset = 50;                                           // отступ от т
double price;                                             // цена срабатыв
double point=SymbolInfoDouble(_Symbol,SYMBOL_POINT);       // размер пункта
int digits=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);       // кол-во знаков
//-- проверка типа операции
if(orderType==ORDER_TYPE_BUY_LIMIT)
{
    request.type =ORDER_TYPE_BUY_LIMIT;                      // тип ордера
    price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point; // цена для от
    request.price =NormalizeDouble(price,digits);           // нормализован
}
else if(orderType==ORDER_TYPE_SELL_LIMIT)
{
    request.type =ORDER_TYPE_SELL_LIMIT;                     // тип ордера
    price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point; // цена для от
    request.price =NormalizeDouble(price,digits);           // нормализован
}
else if(orderType==ORDER_TYPE_BUY_STOP)
{
    request.type =ORDER_TYPE_BUY_STOP;                        // тип ордера
    price =SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point; // цена для от
    request.price=NormalizeDouble(price,digits);             // нормализован
}
else if(orderType==ORDER_TYPE_SELL_STOP)
{
    request.type =ORDER_TYPE_SELL_STOP;                      // тип ордера
    price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point; // цена для от
    request.price =NormalizeDouble(price,digits);           // нормализован
}
else Alert("Этот пример только для установки отложенных ордеров"); // если выбран
//-- отправка запроса
if(!OrderSend(request,result))
    PrintFormat("OrderSend error %d",GetLastError());          // если отправка
//-- информация об операции
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}

```

Пример торговой операции **TRADE\_ACTION\_SLTP** для изменения значений Stop Loss и Take Profit у открытой позиции:

```
#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Модификация Stop Loss и Take Profit позиции |
//+-----+
void OnStart()
{
//--- объявление запроса и результата
MqlTradeRequest request;
MqlTradeResult result;
int total=PositionsTotal(); // количество открытых позиций
//--- перебор всех открытых позиций
for(int i=0; i<total; i++)
{
//--- параметры ордера
ulong position_ticket=PositionGetTicket(i); // тикет позиции
string position_symbol=PositionGetString(POSITION_SYMBOL); // символ
int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS); // количество
ulong magic=PositionGetInteger(POSITION_MAGIC); // MagicNumber позиции
double volume=PositionGetDouble(POSITION_VOLUME); // объем позиции
double sl=PositionGetDouble(POSITION_SL); // Stop Loss позиции
double tp=PositionGetDouble(POSITION_TP); // Take Profit позиции
ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
//--- вывод информации о позиции
PrintFormat("#%I64u %s %s %.2f %s sl: %s tp: %s [%I64d]",
position_ticket,
position_symbol,
EnumToString(type),
volume,
DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
DoubleToString(sl,digits),
DoubleToString(tp,digits),
magic);
//--- если MagicNumber совпадает, Stop Loss и Take Profit не заданы
if(magic==EXPERT_MAGIC && sl==0 && tp==0)
{
```

```

//--- вычисление текущих ценовых уровней
double price=PositionGetDouble(POSITION_PRICE_OPEN);
double bid=SymbolInfoDouble(position_symbol,SYMBOL_BID);
double ask=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
int stop_level=(int)SymbolInfoInteger(position_symbol,SYMBOL_TRADE_STOPS_LEVEL);
double price_level;
//--- если уровень минимально допустимого отступа в пунктах от текущей цены <=0
if(stop_level<=0)
    stop_level=150; // зададим отступ в 150 пунктов от текущей цены закрытия
else
    stop_level+=50; // уровень отступа возьмем равным (SYMBOL_TRADE_STOPS_LEVEL)

//--- вычисление и округление значений Stop Loss и Take Profit
price_level=stop_level*SymbolInfoDouble(position_symbol,SYMBOL_POINT);
if(type==POSITION_TYPE_BUY)
{
    sl=NormalizeDouble(bid-price_level,digits);
    tp=NormalizeDouble(ask+price_level,digits);
}
else
{
    sl=NormalizeDouble(ask+price_level,digits);
    tp=NormalizeDouble(bid-price_level,digits);
}
//--- обнуление значений запроса и результата
ZeroMemory(request);
ZeroMemory(result);
//--- установка параметров операции
request.action =TRADE_ACTION_SLTP; // тип торговой операции
request.position=position_ticket; // тикет позиции
request.symbol=position_symbol; // символ
request.sl =sl; // Stop Loss позиции
request.tp =tp; // Take Profit позиции
request.magic=EXPERT_MAGIC; // MagicNumber позиции
//--- вывод информации о модификации
PrintFormat("Modify #%"I64d "s %s",position_ticket,position_symbol,EnumToString(request.action));
//--- отправка запроса
if(!OrderSend(request,result))
    PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
//--- информация об операции
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
}

```

Пример торговой операции **TRADE\_ACTION MODIFY** для модификации уровней цен отложенных ордеров:

## Константы, перечисления и структуры

```

#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Модификация отложенных ордеров |
//+-----+
void OnStart()
{
//-- объявление и инициализация запроса и результата
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
    int total=OrdersTotal(); // количество установленных отложенных ордеров
//--- перебор всех установленных отложенных ордеров
    for(int i=0; i<total; i++)
    {
//--- параметры ордера
        ulong order_ticket=OrderGetTicket(i); // тикет ордера
        string order_symbol=Symbol(); // символ
        int digits=(int)SymbolInfoInteger(order_symbol,SYMBOL_DIGITS); // количество знаков
        ulong magic=OrderGetInteger(ORDER_MAGIC); // MagicNumber
        double volume=OrderGetDouble(ORDER_VOLUME_CURRENT); // текущий объем
        double sl=OrderGetDouble(ORDER_SL); // текущий Stop Loss
        double tp=OrderGetDouble(ORDER_TP); // текущий Take Profit
        ENUM_ORDER_TYPE type=(ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE); // тип ордера
        int offset = 50; // отступ от цены
        double price; // цена срабатывания
        double point=SymbolInfoDouble(order_symbol,SYMBOL_POINT); // размер пункта
//--- вывод информации об ордере
        PrintFormat("#%I64u %s %s %.2f %s sl: %s tp: %s [%I64d]", // формат вывода
            order_ticket,
            order_symbol,
            EnumToString(type),
            volume,
            DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
            DoubleToString(sl,digits),
            DoubleToString(tp,digits),
            magic);
//--- если MagicNumber совпадает, Stop Loss и Take Profit не заданы
        if(magic==EXPERT_MAGIC && sl==0 && tp==0)
        {
            request.action=TRADE_ACTION MODIFY; // тип торговой операции
            request.order = OrderGetTicket(i); // тикет ордера
            request.symbol =Symbol(); // символ
            request.deviation=5; // допустимое смещение
//--- установка уровня цены, тейк-профит и стоп-лосс ордера в зависимости от типа
            if(type==ORDER_TYPE_BUY_LIMIT)
            {
                price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
                request.tp = NormalizeDouble(price+offset*point,digits);
                request.sl = NormalizeDouble(price-offset*point,digits);
                request.price =NormalizeDouble(price,digits); // нормализация
            }
            else if(type==ORDER_TYPE_SELL_LIMIT)
            {
                price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
                request.tp = NormalizeDouble(price-offset*point,digits);
                request.sl = NormalizeDouble(price+offset*point,digits);
                request.price =NormalizeDouble(price,digits); // нормализация
            }
            else if(type==ORDER_TYPE_BUY_STOP)
            {
                price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
                request.tp = NormalizeDouble(price+offset*point,digits);
            }
        }
    }
}

```

```

request.sl = NormalizeDouble(price-offset*point,digits);
request.price      =NormalizeDouble(price,digits);                                // норма
}
else if(type==ORDER_TYPE_SELL_STOP)
{
price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
request.tp = NormalizeDouble(price-offset*point,digits);
request.sl = NormalizeDouble(price+offset*point,digits);
request.price      =NormalizeDouble(price,digits);                                // норма
}
//--- отправка запроса
if(!OrderSend(request,result))
    PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
//--- информация об операции
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,
//--- обнуление значений запроса и результата
ZeroMemory(request);
ZeroMemory(result);
}
}

```

Пример торговой операции `TRADE_ACTION_REMOVE` для удаления отложенных ордеров:

```
#define EXPERT_MAGIC 123456 // MagicNumber эксперта

//+-----+
//| Удаление отложенных ордеров |
//+-----+

void OnStart()
{
    //-- объявление и инициализация запроса и результата
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
    int total=OrdersTotal(); // количество установленных отложенных ордеров
//--- перебор всех установленных отложенных ордеров
    for(int i=total-1; i>=0; i--)
    {
        ulong order_ticket=OrderGetTicket(i); // тикет ордера
        ulong magic=OrderGetInteger(ORDER_MAGIC); // MagicNumber ордера
        //--- если MagicNumber совпадает
        if(magic==EXPERT_MAGIC)
        {
            //--- обнуление значений запроса и результата
            ZeroMemory(request);
            ZeroMemory(result);
            //--- установка параметров операции
            request.action=TRADE_ACTION_REMOVE; // тип торговой операции
            request.order = order_ticket; // тикет ордера
            //--- отправка запроса
            if(!OrderSend(request,result))
                PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
            //--- информация об операции
            PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,
        }
    }
}
```

Пример торговой операции **TRADE\_ACTION\_CLOSE\_BY** для закрытия позиций встречными:

```

#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Закрытие всех позиций встречными |+
//+-----+
void OnStart()
{
//--- объявление запроса и результата
    MqlTradeRequest request;
    MqlTradeResult result;
    int total=PositionsTotal(); // количество открытых позиций
//--- перебор всех открытых позиций
    for(int i=total-1; i>=0; i--)
    {
//--- параметры ордера
        ulong position_ticket=PositionGetTicket(i);
        string position_symbol=PositionGetString(POSITION_SYMBOL);
        int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
        ulong magic=PositionGetInteger(POSITION_MAGIC);
        double volume=PositionGetDouble(POSITION_VOLUME);
        double sl=PositionGetDouble(POSITION_SL);
        double tp=PositionGetDouble(POSITION_TP);
        ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
//--- вывод информации о позиции
        PrintFormat("#%I64u %s %s %.2f %s sl: %s tp: %s [%I64d]",
                    position_ticket,
                    position_symbol,
                    EnumToString(type),
                    volume,
                    DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                    DoubleToString(sl,digits),
                    DoubleToString(tp,digits),
                    magic);
//--- если MagicNumber совпадает
        if(magic==EXPERT_MAGIC)
        {
            for(int j=0; j<i; j++)
            {
                string symbol=PositionGetSymbol(j); // символ новой позиции
//--- если символы новой и искомой позиций совпадают
                if(symbol==position_symbol && PositionGetInteger(POSITION_MAGIC)==EXPERT_M
                {
//--- установка типа встречной позиции
                    ENUM_POSITION_TYPE type_by=(ENUM_POSITION_TYPE)PositionGetInteger(POSIT
//--- выход, если типы исходной и встречной позиций совпадают
                    if(type==type_by)
                        continue;
//--- обнуление значений запроса и результата
                    ZeroMemory(request);
                    ZeroMemory(result);
//--- установка параметров операции
                    request.action=TRADE_ACTION_CLOSE_BY; // тип т
                    request.position=position_ticket; // тикет
                    request.position_by=PositionGetInteger(POSITION_TICKET); // тикет
//request.symbol =position_symbol;
                    request.magic=EXPERT_MAGIC; // MagicN
//--- вывод информации о закрытии встречной
                    PrintFormat("Close #I64d %s %s by #I64d",position_ticket,position_sy
//--- отправка запроса
                    if(!OrderSend(request,result))
                        PrintFormat("OrderSend error %d",GetLastError()); // если отправить
}

```

```
//--- информация об операции
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
}
}
}
```

## Типы торговых транзакций

В результате выполнения определенных действий с торговым счетом, его состояние изменяется. К таким действиям относятся:

- Отсылка торгового запроса любым MQL5-приложением в клиентском терминале при помощи функций [OrderSend](#) и [OrderSendAsync](#) и его последующее исполнение;
- Отсылка торгового запроса через графический интерфейс терминала и его последующее исполнение;
- Срабатывание отложенных ордеров и стоп-ордеров на сервере;
- Выполнение операций на стороне торгового сервера.

В результате данных действий, для счета выполняются торговые транзакции:

- обработка торгового запроса;
- изменение открытых ордеров;
- изменение истории ордеров;
- изменение истории сделок;
- изменение позиций.

Например, при отсылке рыночного ордера на покупку, он обрабатывается, для счета создается соответствующий ордер на покупку, происходит исполнение ордера, его удаление из списка открытых, добавление в историю ордеров, далее добавляется соответствующая сделка в историю и создается новая позиция. Все эти действия являются торговыми транзакциями.

Для того чтобы программист мог отслеживать действия, осуществляемые относительно торгового счета, предусмотрена функция [OnTradeTransaction](#). При помощи данного обработчика в MQL5-приложении можно получать торговые транзакции, примененные к счету. Описание торговой транзакции передается в первом параметре `OnTradeTransaction` при помощи структуры [MqlTradeTransaction](#).

Тип торговой транзакции передается в параметре `type` структуры `MqlTradeTransaction`. Возможные типы торговых транзакций описываются следующим перечислением:

### ENUM\_TRADE\_TRANSACTION\_TYPE

Идентификатор	Описание
TRADE_TRANSACTION_ORDER_ADD	Добавление нового открытого ордера.
TRADE_TRANSACTION_ORDER_UPDATE	Изменение открытого ордера. К данным изменениям относятся не только явные изменения со стороны клиентского терминала или торгового сервера, но также и изменение его состояния при выставлении (например, переход из состояния <a href="#">ORDER_STATE_STARTED</a> в <a href="#">ORDER_STATE_PLACED</a> или из <a href="#">ORDER_STATE_PLACED</a> в <a href="#">ORDER_STATE_PARTIAL</a> и т.д.).
TRADE_TRANSACTION_ORDER_DELETE	Удаление ордера из списка открытых. Ордер может быть удален из открытых в результате

	выставления соответствующего запроса либо в результате исполнения (заливки) и переноса в историю.
TRADE_TRANSACTION_DEAL_ADD	Добавление сделки в историю. Осуществляется в результате исполнения ордера или проведения операций с балансом счета.
TRADE_TRANSACTION DEAL_UPDATE	Изменение сделки в истории. Возможны ситуации, когда ранее исполненная сделка изменяется на сервере. Например, сделка была изменена во внешней торговой системе (бирже), куда она была выведена брокером.
TRADE_TRANSACTION DEAL_DELETE	Удаление сделки из истории. Возможны ситуации, когда ранее исполненная сделка удаляется на сервере. Например, сделка была удалена во внешней торговой системе (бирже), куда она была выведена брокером.
TRADE_TRANSACTION_HISTORY_ADD	Добавление ордера в историю в результате исполнения или отмены.
TRADE_TRANSACTION_HISTORY_UPDATE	Изменение ордера, находящегося в истории ордеров. Данный тип предусмотрен для расширения функциональности на стороне торгового сервера.
TRADE_TRANSACTION_HISTORY_DELETE	Удаление ордера из истории ордеров. Данный тип предусмотрен для расширения функциональности на стороне торгового сервера.
TRADE_TRANSACTION_POSITION	Изменение позиции, не связанное с исполнением сделки. Данный тип транзакции свидетельствует именно о том, что позиция была изменена на стороне торгового сервера. У позиции может быть изменен объем, цена открытия, а также уровни Stop Loss и Take Profit. Информация об изменениях передается в структуре <a href="#">MqlTradeTransaction</a> через обработчик OnTradeTransaction. Изменение позиции (добавление, изменение или ликвидация) в результате совершения сделки не влечет за собой появление транзакции TRADE_TRANSACTION_POSITION.
TRADE_TRANSACTION_REQUEST	Уведомление о том, что торговый запрос обработан сервером, и результат его обработки получен. Для транзакций данного типа в структуре <a href="#">MqlTradeTransaction</a> необходимо анализировать только одно поле - type (тип транзакции). Для получения

дополнительной информации необходимо анализировать второй и третий параметры функции [OnTradeTransaction](#) (request и result).

В зависимости от типа торговой транзакции, в структуре MqlTradeTransaction, описывающей ее, заполняются различные параметры. Подробное описание передаваемых данных приведено в разделе "[Структура торговой транзакции](#)".

#### Смотри также

[Структура торговой транзакции](#), [OnTradeTransaction](#)

## Виды заявок в стакане цен

Для биржевых инструментов доступно окно "Стакан цен", в котором можно посмотреть текущие заявки на покупку и продажу. Для каждой заявки указано желаемое направление торговой операции, требуемый объем и запрашиваемая цена.

Для получения информации о текущем состоянии стакана цен средствами языка MQL5 предназначена функция [MarketBookGet\(\)](#), которая помещает "снимок стакана" в массив структур [MqlBookInfo](#). Каждый элемент этого массива в поле *type* содержит информацию о направлении заявки - это значение из перечисления ENUM\_BOOK\_TYPE.

### ENUM\_BOOK\_TYPE

Идентификатор	Описание
BOOK_TYPE_SELL	Заявка на продажу
BOOK_TYPE_BUY	Заявка на покупку
BOOK_TYPE_SELL_MARKET	Заявка на продажу по рыночной цене
BOOK_TYPE_BUY_MARKET	Заявка на покупку по рыночной цене

### Смотри также

[Структуры и классы](#), [Структура стакана цен](#), [Типы торговых операций](#), [Получение рыночной информации](#)

## Свойства сигналов

Значения перечислений для работы с торговыми сигналами и настройками их копирования.

Перечисления свойств типа [double](#) торговых сигналов:

### ENUM\_SIGNAL\_BASE\_DOUBLE

Константа	Описание
SIGNAL_BASE_BALANCE	Баланс счета
SIGNAL_BASE_EQUITY	Средства на счете
SIGNAL_BASE_GAIN	Прирост счета в процентах
SIGNAL_BASE_MAX_DRAWDOWN	Максимальная просадка
SIGNAL_BASE_PRICE	Цена подписки на сигнал
SIGNAL_BASE_ROI	Значение ROI (Return on Investment) сигнала в %

Перечисления свойств типа [integer](#) торговых сигналов:

### ENUM\_SIGNAL\_BASE\_INTEGER

Константа	Описание
SIGNAL_BASE_DATE_PUBLISHED	Дата публикации сигнала (когда стал доступен для подписки)
SIGNAL_BASE_DATE_STARTED	Дата начала мониторинга сигнала
SIGNAL_BASE_DATE_UPDATED	Дата последнего обновления торговой статистики сигнала
SIGNAL_BASE_ID	ID сигнала
SIGNAL_BASE_LEVERAGE	Плечо торгового счета
SIGNAL_BASE_PIPS	Результат торговли в пipsах
SIGNAL_BASE_RATING	Позиция в рейтинге сигналов
SIGNAL_BASE_SUBSCRIBERS	Количество подписчиков
SIGNAL_BASE_TRADES	Количество трейдов
SIGNAL_BASE_TRADE_MODE	Тип счета (0-реальный счет, 1-демо-счет, 2-конкурсный счет)

Перечисления свойств типа [string](#) торговых сигналов:

### ENUM\_SIGNAL\_BASE\_STRING

Константа	Описание

SIGNAL_BASE_AUTHOR_LOGIN	Логин автора сигнала
SIGNAL_BASE_BROKER	Наименование брокера (компании)
SIGNAL_BASE_BROKER_SERVER	Сервер брокера
SIGNAL_BASE_NAME	Имя сигнала
SIGNAL_BASE_CURRENCY	Валюта счета сигнала

Перечисления свойств типа [double](#) настроек копирования торговых сигналов:

#### ENUM\_SIGNAL\_INFO\_DOUBLE

Константа	Описание
SIGNAL_INFO_EQUITY_LIMIT	Процент для конвертации объема сделки
SIGNAL_INFO_SLIPPAGE	Величина проскальзывания, с которым выставляются рыночные ордера при синхронизации позиций и копировании сделок
SIGNAL_INFO_VOLUME_PERCENT	Значение ограничения по средствам для сигнала, r/o

Перечисления свойств типа [integer](#) настроек копирования торговых сигналов:

#### ENUM\_SIGNAL\_INFO\_INTEGER

Константа	Описание
SIGNAL_INFO_CONFIRMATIONS_DISABLED	Флаг разрешения синхронизации без показа диалога подтверждения
SIGNAL_INFO_COPY_SLTP	Флаг копирования Stop Loss и Take Profit
SIGNAL_INFO_DEPOSIT_PERCENT	Ограничения по депозиту (в %)
SIGNAL_INFO_ID	id сигнала, r/o
SIGNAL_INFO_SUBSCRIPTION_ENABLED	Флаг разрешения на копирование сделок по подписке
SIGNAL_INFO_TERMS_AGREE	Флаг согласия с условиями использования сервиса "Сигналы", r/o

Перечисления свойств типа [string](#) настроек копирования торговых сигналов:

#### ENUM\_SIGNAL\_INFO\_STRING

Константа	Описание
SIGNAL_INFO_NAME	Имя сигнала, r/o

Смотри также

[Управление сигналами](#)

## Именованные константы

Все используемые в языке MQL5 константы можно разбить на следующие группы:

- [Предопределенные макроподстановки](#) - значения подставляются на этапе компиляции;
- [Математические константы](#) - значения некоторых математических выражений;
- [Константы числовых типов](#) - ограничения, накладываемые на простые типы;
- [Причины деинициализации](#) - описание причин деинициализации;
- [Проверка указателя объекта](#) - перечисление типов указателей, возвращаемых функцией `CheckPointer()` ;
- [Прочие константы](#) - все, что не вошло в остальные группы констант.

## Предопределенные макроподстановки

Для облегчения отладки и получения информации о работе mq5-программы введены специальные константы-макросы, значения которых устанавливается в момент компиляции. Самый простой путь использования этих констант - вывод значений с помощью функции [Print\(\)](#), как показано в примере.

Константа	Описание
<code>__DATE__</code>	Дата компиляции файла без времени (часы, минуты и секунды равны 0)
<code>__DATETIME__</code>	Дата и время компиляции файла
<code>__LINE__</code>	Номер строки в исходном коде, на которой расположен данный макрос
<code>__FILE__</code>	Имя текущего компилируемого файла
<code>__PATH__</code>	Абсолютный путь к текущему компилируемому файлу
<code>__FUNCTION__</code>	Имя функции, в теле которой расположен макрос
<code>__FUNCSIG__</code>	Сигнатура функции, в теле которой расположен макрос. Вывод в лог полного описания функции с типами параметров может пригодиться при идентификации <a href="#">перегруженных функций</a>
<code>__MQLBUILD__, __MQL5BUILD__</code>	Номер билда компилятора

**Пример:**

```
#property copyright "Copyright © 2009, MetaQuotes Software Corp."
#property link      "https://www.metaquotes.net"

//+-----+
//| Expert initialization function
//+-----+

void OnInit()
{
    //--- пример вывода информации при инициализации советника
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__);
    //--- установка интервала между событиями таймера
    EventSetTimer(5);
    //---
}

//+-----+
//| Expert deinitialization function
//+-----+

void OnDeinit(const int reason)
{
    //--- пример вывода информации при деинициализации советника
}
```

```
Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__);
//---
{
}
//+-----+
//| Expert tick function
//+-----+
void OnTick()
{
    //--- вывод информации при поступлении тика
    Print(" __MQLBUILD__ = ", __MQLBUILD__, " __FILE__ = ", __FILE__);
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__);
    test1(__FUNCTION__);
    test2();
//---
}
//+-----+
//| test1
//+-----+
void test1(string par)
{
    //--- вывод информации внутри функции
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__, " par=",par);
}
//+-----+
//| test2
//+-----+
void test2()
{
    //--- вывод информации внутри функции
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__);
}
//+-----+
//| OnTimer event handler
//+-----+
void OnTimer()
{
    //---
    Print(" __FUNCTION__ = ", __FUNCTION__, " __LINE__ = ", __LINE__);
    test1(__FUNCTION__);
}
```

## Математические константы

Для некоторых математических выражений зарезервированы специальные константы, содержащие значения. Эти константы можно использовать в любом месте mqfl5-программы вместо вычисления их значения с помощью [математических функций](#).

Константа	Описание	Значение
M_E	e	2.71828182845904523536
M_LOG2E	log2(e)	1.44269504088896340736
M_LOG10E	log10(e)	0.434294481903251827651
M_LN2	ln(2)	0.693147180559945309417
M_LN10	ln(10)	2.30258509299404568402
M_PI	pi	3.14159265358979323846
M_PI_2	pi/2	1.57079632679489661923
M_PI_4	pi/4	0.785398163397448309616
M_1_PI	1/pi	0.318309886183790671538
M_2_PI	2/pi	0.636619772367581343076
M_2_SQRTPI	2/sqrt(pi)	1.12837916709551257390
M_SQRT2	sqrt(2)	1.41421356237309504880
M_SQRT1_2	1/sqrt(2)	0.707106781186547524401

Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- выводим значения констант
Print("M_E = ",DoubleToString(M_E,16));
Print("M_LOG2E = ",DoubleToString(M_LOG2E,16));
Print("M_LOG10E = ",DoubleToString(M_LOG10E,16));
Print("M_LN2 = ",DoubleToString(M_LN2,16));
Print("M_LN10 = ",DoubleToString(M_LN10,16));
Print("M_PI = ",DoubleToString(M_PI,16));
Print("M_PI_2 = ",DoubleToString(M_PI_2,16));
Print("M_PI_4 = ",DoubleToString(M_PI_4,16));
Print("M_1_PI = ",DoubleToString(M_1_PI,16));
Print("M_2_PI = ",DoubleToString(M_2_PI,16));
Print("M_2_SQRTPI = ",DoubleToString(M_2_SQRTPI,16));
Print("M_SQRT2 = ",DoubleToString(M_SQRT2,16));
Print("M_SQRT1_2 = ",DoubleToString(M_SQRT1_2,16));
```

{}

## Константы числовых типов данных

Каждый простой числовой тип предназначен для определенного круга задач и позволяет оптимизировать работу mqfl5-программы при правильном применении. Для лучшей читаемости кода и правильной обработки результатов вычислений введены константы, которые позволяют получить информацию об ограничениях, накладываемых на тот или иной тип простых данных.

Константа	Описание	Значение
CHAR_MIN	Минимальное значение, которое может быть представлено типом char	-128
CHAR_MAX	Максимальное значение, которое может быть представлено типом char	127
UCHAR_MAX	Максимальное значение, которое может быть представлено типом uchar	255
SHORT_MIN	Минимальное значение, которое может быть представлено типом short	-32768
SHORT_MAX	Максимальное значение, которое может быть представлено типом short	32767
USHORT_MAX	Максимальное значение, которое может быть представлено типом ushort	65535
INT_MIN	Минимальное значение, которое может быть представлено типом int	-2147483648
INT_MAX	Максимальное значение, которое может быть представлено типом int	2147483647
UINT_MAX	Максимальное значение, которое может быть представлено типом uint	4294967295
LONG_MIN	Минимальное значение, которое может быть представлено типом long	-9223372036854775808
LONG_MAX	Максимальное значение, которое может быть представлено типом long	9223372036854775807
ULONG_MAX	Максимальное значение, которое может быть представлено типом ulong	18446744073709551615

DBL_MIN	Минимальное положительное значение, которое может быть представлено типом double	2.2250738585072014e-308
DBL_MAX	Максимальное значение, которое может быть представлено типом double	1.7976931348623158e+308
DBL_EPSILON	Наименьшее число для которого выполняется условие $1.0 + \text{DBL\_EPSILON} \neq 1.0$	2.2204460492503131e-016
DBL_DIG	Количество значимых десятичных знаков	15
DBL_MANT_DIG	Количество битов в мантиссе	53
DBL_MAX_10_EXP	Максимальное десятичное значение степени экспоненты	308
DBL_MAX_EXP	Максимальное двоичное значение степени экспоненты	1024
DBL_MIN_10_EXP	Минимальное десятичное значение степени экспоненты	(-307)
DBL_MIN_EXP	Минимальное двоичное значение степени экспоненты	(-1021)
FLOAT_MIN	Минимальное положительное значение, которое может быть представлено типом float	1.175494351e-38
FLOAT_MAX	Максимальное значение, которое может быть представлено типом float	3.402823466e+38
FLOAT_EPSILON	Наименьшее число для которого выполняется условие $1.0 + \text{FLOAT\_EPSILON} \neq 1.0$	1.192092896e-07
FLOAT_DIG	Количество значимых десятичных знаков	6
FLOAT_MANT_DIG	Количество битов в мантиссе	24
FLOAT_MAX_10_EXP	Максимальное десятичное значение степени экспоненты	38
FLOAT_MAX_EXP	Максимальное двоичное значение степени экспоненты	128

FLT_MIN_10_EXP	Минимальное десятичное значение степени экспоненты	-37
FLT_MIN_EXP	Минимальное двоичное значение степени экспоненты	(-125)

**Пример:**

```
void OnStart()
{
//--- выведем значения констант
printf("CHAR_MIN = %d",CHAR_MIN);
printf("CHAR_MAX = %d",CHAR_MAX);
printf("UCHAR_MAX = %d",UCHAR_MAX);
printf("SHORT_MIN = %d",SHORT_MIN);
printf("SHORT_MAX = %d",SHORT_MAX);
printf("USHORT_MAX = %d",USHORT_MAX);
printf("INT_MIN = %d",INT_MIN);
printf("INT_MAX = %d",INT_MAX);
printf("UINT_MAX = %u",UINT_MAX);
printf("LONG_MIN = %I64d",LONG_MIN);
printf("LONG_MAX = %I64d",LONG_MAX);
printf("ULONG_MAX = %I64u",ULONG_MAX);
printf("EMPTY_VALUE = %.16e",EMPTY_VALUE);
printf("DBL_MIN = %.16e",DBL_MIN);
printf("DBL_MAX = %.16e",DBL_MAX);
printf("DBL_EPSILON = %.16e",DBL_EPSILON);
printf("DBL_DIG = %d",DBL_DIG);
printf("DBL_MANT_DIG = %d",DBL_MANT_DIG);
printf("DBL_MAX_10_EXP = %d",DBL_MAX_10_EXP);
printf("DBL_MAX_EXP = %d",DBL_MAX_EXP);
printf("DBL_MIN_10_EXP = %d",DBL_MIN_10_EXP);
printf("DBL_MIN_EXP = %d",DBL_MIN_EXP);
printf("FLT_MIN = %.8e",FLT_MIN);
printf("FLT_MAX = %.8e",FLT_MAX);
printf("FLT_EPSILON = %.8e",FLT_EPSILON);
}
```

## Причины деинициализации

Коды причины деинициализации [эксперта](#), возвращаемые функцией [UninitializeReason\(\)](#). Могут иметь любые из следующих значений:

Константа	Значение	Описание
REASON_PROGRAM	0	Эксперт прекратил свою работу, вызвав функцию <a href="#">ExpertRemove()</a>
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 либо произошло переподключение к торговому серверу вследствие изменения настроек счета
REASON_TEMPLATE	7	Применен другой шаблон графика
REASON_INITFAILED	8	Признак того, что обработчик <a href="#">OnInit()</a> вернул ненулевое значение
REASON_CLOSE	9	Терминал был закрыт

Код причины деинициализации передается также в качестве параметра предопределенной функции [OnDeinit\(const int reason\)](#).

**Пример:**

```
//+-----+
//| get text description
//+-----+
string getUninitReasonText(int reasonCode)
{
    string text="";
//---
    switch(reasonCode)
    {
```

```
case REASON_ACCOUNT:
    text="Account was changed";break;
case REASON_CHARTCHANGE:
    text="Symbol or timeframe was changed";break;
case REASON_CHARTCLOSE:
    text="Chart was closed";break;
case REASON_PARAMETERS:
    text="Input-parameter was changed";break;
case REASON_RECOMPILE:
    text="Program "+__FILE__+" was recompiled";break;
case REASON_REMOVE:
    text="Program "+__FILE__+" was removed from chart";break;
case REASON_TEMPLATE:
    text="New template was applied to chart";break;
default:text="Another reason";
}

//---
return text;
}
//----------------------------------------------------------------------------------------------------------------+
//| Expert deinitialization function
//----------------------------------------------------------------------------------------------------------------+
void OnDeinit(const int reason)
{
//--- Первый способ получить код причины деинициализации
Print(__FUNCTION__,"_Код причины деинициализации = ",reason);
//--- Второй способ получить код причины деинициализации
Print(__FUNCTION__,"_UninitReason = ",getUninitReasonText(_UninitReason));
}
```

## Проверка указателя объекта

Для проверки типа [указателя объекта](#) предназначена функция [CheckPointer\(\)](#), которая возвращает значение из перечисления ENUM\_POINTER\_TYPE. В случае использования некорректного указателя исполнение программы будет немедленно прервано.

Объекты, созданные оператором [new](#), имеют тип POINTER\_DYNAMIC. Только для таких указателей можно и нужно использовать [оператор уничтожения delete\(\)](#).

Все остальные указатели имеют тип POINTER\_AUTOMATIC, что означает, что данный объект был создан автоматически средой исполнения mql5-программы. Такие объекты уничтожаются после использования также автоматически.

### ENUM\_POINTER\_TYPE

Константа	Описание
POINTER_INVALID	Некорректный указатель
POINTER_DYNAMIC	Указатель объекта, созданного оператором <a href="#">new</a>
POINTER_AUTOMATIC	Указатель любого объекта, созданного автоматически (без использования new())

#### Смотри также

[Ошибки выполнения](#), [Оператор уничтожения объекта delete](#), [CheckPointer\(\)](#)

## Прочие константы

Константа CLR\_NONE служит для указания отсутствия цвета, то есть [графический объект](#) или [графическая серия](#) индикатора не будут отображены. Эта константа не вошла в список констант с наименованиями [Web-цветов](#), но может применяться везде, где требуется указание цвета.

Константа INVALID\_HANDLE может использоваться при проверке файловых хэндов (см. функции [FileOpen\(\)](#) и [FileFindFirst\(\)](#)).

Константа	Описание	Значение
CHARTS_MAX	Максимально возможное количество одновременно открытых графиков в терминале	100
clrNONE	Отсутствие цвета	-1
EMPTY_VALUE	Пустое значение в индикаторном буфере	DBL_MAX
INVALID_HANDLE	Некорректный хэндл	-1
IS_DEBUG_MODE	Признак работы mq5-программы в режиме отладки	в режиме отладки не равно нулю, в противном случае 0
IS_PROFILE_MODE	Признак работы mq5-программы в режиме профилирования	в режиме профилирования не равно нулю, в противном случае 0
NULL	Ноль любого типа	0
WHOLE_ARRAY	Означает количество элементов, оставшееся до конца массива, то есть, будет обработан весь массив	-1
WRONG_VALUE	Константа может неявно <a href="#">приводиться</a> к типу <a href="#">любого перечисления</a> .	-1

Константа EMPTY\_VALUE обычно соответствует тем значениям индикаторов, которые не отрисовываются на графике. Например, для встроенного индикатора Standard Deviation с периодом 20 не выводится на график линия для первых в истории 19 баров. Если создать хэндл этого индикатора с помощью функции [iStdDev\(\)](#) и скопировать в массив значения индикатора для этих баров через [CopyBuffer\(\)](#), то эти значения как раз и будут равны EMPTY\_VALUE.

Можно самостоятельно указать в [пользовательском индикаторе](#) собственное пустое значение индикатора, при котором не должна производиться отрисовка на графике. Для этого используйте функция [PlotIndexSetDouble\(\)](#) с модификатором [PLOT\\_EMPTY\\_VALUE](#).

Константа [NULL](#) может быть присвоена переменной любого простого типа или указателю на объект структуры или класса. Присвоение NULL строковой переменной означает полную деинициализацию этой переменной.

Константа `WRONG_VALUE` предназначена для тех случаев, когда требуется вернуть значение [перечисления](#), и это должно быть неверное значение. Например, нужно сообщить, что возвращаемое значение является значением этого перечисления. Приведем в качестве иллюстрации некую функцию `CheckLineStyle()`, которая возвращает стиль линии для объекта, указанного по имени. Если при запросе стиля функцией `ObjectGetInteger()` результатом будет `true`, то вернется значение перечисления [ENUM\\_LINE\\_STYLE](#), иначе возвращается `WRONG_VALUE`.

```
void OnStart()
{
    if(CheckLineStyle("MyChartObject") == WRONG_VALUE)
        printf("Error line style getting.");
}

//+-----+
//| возвращает стиль линии для объекта, указанного по имени |
//+-----+
ENUM_LINE_STYLE CheckLineStyle(string name)
{
    long style;
//---
    if(ObjectGetInteger(0, name, OBJPROP_STYLE, 0, style))
        return((ENUM_LINE_STYLE)style);
    else
        return(WRONG_VALUE);
}
```

Константа `WHOLE_ARRAY` предназначена для функций, которые требуют указания количества элементов в обрабатываемых массивах:

- [ArrayCopy\(\)](#);
- [ArrayMinimum\(\)](#);
- [ArrayMaximum\(\)](#);
- [FileReadArray\(\)](#);
- [FileWriteArray\(\)](#).

Если требуется указать, что необходимо обработать все значения массива с указанной позиции и до конца, то достаточно указать значение `WHOLE_ARRAY`.

Константа `IS_PROFILE_MODE` позволяет изменить работу программы для корректного сбора информации в режиме профилирования. Профилирование позволяет замерить время выполнения отдельных фрагментов программы (обычно это функции), а также подсчитать количество таких вызовов. Для корректного получения информации о времени выполнения в режиме профилировки можно отключить вызовы функции `Sleep()` как в примере:

```
---- Sleep может сильно повлиять (искажить) на результат профилировки
if(!IS_PROFILE_MODE) Sleep(100); // запрещаем вызов Sleep() в режиме профилировки
```

Значение константы `IS_PROFILE_MODE` задается компилятором в момент компиляции, и в обычном режиме выставляется равным нулю. При запуске программы в режиме профилирования производится специальная компиляция, и в этом случае вместо `IS_PROFILE_MODE` подставляется значение отличное от нуля.

Константа IS\_DEBUG\_MODE пригодится в тех случаях, когда необходимо немного изменить работу MQL5-программы в режиме отладки. Например, в режиме отладки может потребоваться выводить дополнительную отладочную информацию в лог терминала или создавать вспомогательные графические объекты на графике.

Приведенный ниже пример создает объект Label и задает ее описание и цвет в зависимости от того, в каком режиме выполняется скрипт. Для того чтобы запустить скрипт в режиме отладки из MetaEditor, нажмите клавишу F5. Если запустить скрипта из окна навигатора в терминале, то цвет и текст объекта Label будут другими.

### Пример:

```
//+-----+
//| Check_DEBUG_MODE.mq5 | Copyright © 2009, MetaQuotes Software Corp. |
//| https://www.metaquotes.net | |
//+-----+
#property copyright "Copyright © 2009, MetaQuotes Software Corp."
#property link      "https://www.metaquotes.net"
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//---
    string label_name="invisible_label";
    if(ObjectFind(0,label_name)<0)
    {
        Print("Object ",label_name," not found. Error code = ",GetLastError());
        //--- создадим объект Label
        ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
        //--- установим координату X
        ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
        //--- установим координату Y
        ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
        ResetLastError();
        if(IS_DEBUG_MODE) // режим отладки
        {
            //--- выведем сообщение о режиме выполнения скрипта
            ObjectSetString(0,label_name,OBJPROP_TEXT,"DEBUG MODE");
            //--- зададим красный цвет текста
            if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrRed))
                Print("Не удалось установить цвет. Ошибка ",GetLastError());
        }
        else // рабочий режим
        {
            ObjectSetString(0,label_name,OBJPROP_TEXT,"RELEASE MODE");
            //--- зададим невидимый цвет текста
            if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,CLR_NONE))
                Print("Не удалось установить цвет. Ошибка ",GetLastError());
        }
    }
}
```

```

    }
    ChartRedraw();
    DebugBreak(); // здесь произойдет прерывание, если мы в режиме отладки
}
}

```

## Методы шифрования данных

Для указания метода преобразования данных (шифрование и расчет хешей) в функциях [CryptEncode\(\)](#) и [CryptDecode\(\)](#) используется перечисление ENUM\_CRYPT\_METHOD.

### ENUM\_CRYPT\_METHOD

Константа	Описание
CRYPT_BASE64	Шифрование BASE64 (перекодировка)
CRYPT_AES128	Шифрование AES с ключом 128 бит (16 байт)
CRYPT_AES256	Шифрование AES с ключом 256 бит (32 байта)
CRYPT_DES	Шифрование DES с ключом 56 бит (7 байт)
CRYPT_HASH_SHA1	Расчёт HASH SHA1
CRYPT_HASH_SHA256	Расчёт HASH SHA256
CRYPT_HASH_MD5	Расчёт HASH MD5
CRYPT_ARCH_ZIP	ZIP архивирование

Смотри также

[DebugBreak](#), [Информация о запущенной MQL5-программе](#), [CryptEncode\(\)](#), [CryptDecode\(\)](#)

## Структуры данных

В MQL5 существуют 12 предопределенных [структур](#), предназначенные для хранения и передачи служебной информации:

- [MqlDateTime](#) предназначена для представления [даты и времени](#);
- [MqlParam](#) позволяет передавать входные параметры при создании хэндла индикатора с помощью функции [IndicatorCreate\(\)](#);
- [MqlRates](#) предоставляет информацию об [исторических данных](#), содержащих цену, объем и спред;
- [MqlBookInfo](#) для получения информации, отображаемой в [стакане цен](#) (окно котировок);
- [MqlTradeRequest](#) для создания торгового запроса при проведении [торговых операций](#);
- [MqlTradeCheckResult](#) позволяет [проверить](#) подготовленный [торговый запрос](#) перед его [отправкой](#);
- [MqlTradeResult](#) содержит ответ торгового сервера на [торговый запрос](#), отправленный функцией [OrderSend\(\)](#);
- [MqlTradeTransaction](#) содержит описание торговой транзакции;
- [MqlTick](#) предназначена для быстрого получения наиболее востребованной информации о текущих ценах.
- [Структуры Экономического календаря](#) предназначены для получения информации о событиях Экономического календаря, которые поступают в платформу MetaTrader 5 в режиме реального времени. [Функции Экономического календаря](#) позволяют проводить анализ макроэкономических индикаторов сразу же после публикации новых отчетов, так как актуальные значения транслируются напрямую из источника без задержек.

## MqlDateTime

Структура даты содержит в себе восемь полей типа [int](#).

```
struct MqlDateTime
{
    int year;           // год
    int mon;            // месяц
    int day;             // день
    int hour;            // час
    int min;             // минуты
    int sec;             // секунды
    int day_of_week;     // день недели (0-воскресенье, 1-понедельник, ..., 6-суббота)
    int day_of_year;     // порядковый номер в году (1 января имеет номер 0)
};
```

### Примечание

Порядковый номер в году `day_of_year` в високосном году, начиная с марта, будет отличаться от порядкового номера соответствующего дня в невисокосном году.

### Пример:

```
void OnStart()
{
//---
    datetime date1=D'2008.03.01';
    datetime date2=D'2009.03.01';

    MqlDateTime str1,str2;
    TimeToStruct(date1,str1);
    TimeToStruct(date2,str2);
    printf("%02d.%02d.%4d, day of year = %d",str1.day,str1.mon,
           str1.year,str1.day_of_year);
    printf("%02d.%02d.%4d, day of year = %d",str2.day,str2.mon,
           str2.year,str2.day_of_year);
}
/* Результат
01.03.2008, day of year = 60
01.03.2009, day of year = 59
*/
```

### Смотри также

[TimeToStruct](#), [Структуры и классы](#)

## Структура входных параметров индикатора (MqlParam)

Структура MqlParam специально разработана для передачи [входных параметров](#) при создании хэндла [технического индикатора](#) с помощью функции [IndicatorCreate\(\)](#).

```
struct MqlParam
{
    ENUM_DATATYPE type;           // тип входного параметра, значение первого
    long integer_value;          // поля для хранения целочисленного значения
    double double_value;          // поля для хранения значения double или
    string string_value;          // поля для хранения значения строкового типа
};
```

Все входные параметры индикатора передаются в виде массива типа MqlParam, поле *type* каждого элемента этого массива указывает тип данных, передаваемых данным элементом. Сами значения параметров индикатора необходимо предварительно поместить в соответствующие поля каждого элемента (в *integer\_value*, в *double\_value* или в *string\_value*) в зависимости от того, какое значение перечисления [ENUM\\_DATATYPE](#) содержится в поле *type*.

Если функции [IndicatorCreate\(\)](#) третьим параметром в качестве типа индикатора передается значение IND\_CUSTOM, то первый элемент массива входных параметров должен иметь поле *type* со значением TYPE\_STRING из перечисления [ENUM\\_DATATYPE](#), а поле *string\_value* должно содержать имя [пользовательского индикатора](#).

## MqlRates

Структура для хранения информации о ценах, объемах и спреде.

```
struct MqlRates
{
    datetime time;           // время начала периода
    double open;             // цена открытия
    double high;             // наивысшая цена за период
    double low;              // наименьшая цена за период
    double close;            // цена закрытия
    long tick_volume;        // тиковый объем
    int spread;              // спред
    long real_volume;        // биржевой объем
};
```

**Пример:**

```
void OnStart()
{
    MqlRates rates[];
    int copied=CopyRates(NULL,0,0,100,rates);
    if(copied<=0)
        Print("Ошибка копирования ценовых данных ",GetLastError());
    else
        Print("Скопировано ",ArraySize(rates)," баров");
}
```

**Смотри также**

[CopyRates](#), [Доступ к таймсериям](#)

## MqlBookInfo

Структура, предоставляющая информацию в стакане цен.

```
struct MqlBookInfo
{
    ENUM_BOOK_TYPE type;           // тип заявки из перечисления ENUM BOOK TYPE
    double          price;         // цена
    long            volume;        // объем
    double          volume_real;   // объем с повышенной точностью
};
```

### Примечание

Структура MqlBookInfo является предопределенной, поэтому ее объявление и описание не требуется. Чтобы использовать структуру, достаточно объявить переменную данного типа.

Стакан доступен не для всех финансовых инструментов.

### Пример:

```
MqlBookInfo priceArray[];
bool getBook=MarketBookGet(NULL,priceArray);
if(getBook)
{
    int size=ArraySize(priceArray);
    Print("MarketBookInfo по ",Symbol());
}
else
    Print("Не удалось получить содержимое стакана по символу ",Symbol());
```

### Смотри также

[MarketBookAdd](#), [MarketBookRelease](#), [MarketBookGet](#), [Виды заявок в стакане цен](#), [Типы данных](#)

## Структура торгового запроса (MqlTradeRequest)

Взаимодействие клиентского терминала и торгового сервера для проведения операций постановки ордеров производится посредством торговых запросов. Запрос представлен специальной предопределенной [структурой MqlTradeRequest](#), которая содержит все поля, необходимые для заключения торговых сделок. Результат обработки запроса представлен структурой [MqlTradeResult](#).

```
struct MqlTradeRequest
{
    ENUM_TRADE_REQUEST_ACTIONS     action;           // Тип выполняемого действия
    ulong                           magic;            // Штамп эксперта (идентификатор та
    ulong                           order;             // Тикет ордера
    string                          symbol;           // Имя торгового инструмента
    double                          volume;            // Запрашиваемый объем сделки в лотах
    double                          price;             // Цена
    double                          stoplimit;         // Уровень StopLimit ордера
    double                          sl;                // Уровень Stop Loss ордера
    double                          tp;                // Уровень Take Profit ордера
    ulong                           deviation;        // Максимально приемлемое отклонение
    ENUM_ORDER_TYPE                 type;              // Тип ордера
    ENUM_ORDER_TYPE_FILLING         type_filling;     // Тип ордера по исполнению
    ENUM_ORDER_TYPE_TIME            type_time;        // Тип ордера по времени действия
    datetime                        expiration;       // Срок истечения ордера (для ордеров)
    string                          comment;           // Комментарий к ордеру
    ulong                           position;          // Тикет позиции
    ulong                           position_by;       // Тикет встречной позиции
};
```

### Описание полей

Поле	Описание
action	Тип торговой операции. Значение может быть одним из значений перечисления <a href="#">ENUM_TRADE_REQUEST_ACTIONS</a>
magic	Идентификатор эксперта. Позволяет организовать аналитическую обработку торговых ордеров. Каждый эксперт может выставлять свой собственный уникальный идентификатор при отправке торгового запроса
order	Тикет ордера. Требуется для модификации отложенных ордеров
symbol	Имя торгового инструмента, по которому выставляется ордер. Не требуется при

	операциях модификации ордеров и закрытии позиций
volume	Запрашиваемый объем сделки в лотах. Реальное значение объема при открытии сделки будет зависеть от <a href="#">типа ордера по исполнению</a> .
price	Цена, при достижении которой ордер должен быть выполнен. Для рыночных ордеров по инструментам с типом исполнения "Market Execution" ( <a href="#">SYMBOL_TRADE_EXECUTION_MARKET</a> ), имеющих тип <a href="#">TRADE_ACTION_DEAL</a> , указание цены не требуется
stoplimit	Цена, по которой будет выставлен отложенный Limit ордер, при достижении ценой значения price (это условие является обязательным). До этого момента отложенный ордер в торговую систему не выводится
sl	Цена, по которой сработает Stop Loss ордер при движении цены в неблагоприятном направлении
tp	Цена, по которой сработает Take Profit ордер при движении цены в благоприятном направлении
deviation	Максимально приемлемое отклонение от запрашиваемой цены, задаваемое в <a href="#">пунктах</a>
type	Тип ордера. Значение может быть одним из значений перечисления <a href="#">ENUM_ORDER_TYPE</a>
type_filling	Тип ордера по исполнению. Значение может быть одним из значений <a href="#">ENUM_ORDER_TYPE_FILLING</a>
type_time	Тип ордера по истечению. Значение может быть одним из значений <a href="#">ENUM_ORDER_TYPE_TIME</a>
expiration	Срок истечения отложенного ордера (для ордеров типа <a href="#">ORDER_TIME_SPECIFIED</a> )
comment	Комментарий к ордеру
position	Тикет позиции. Следует заполнять при изменении и закрытии позиции для ее однозначной идентификации. Как правило, соответствует тикету ордера, в результате которого позиция была открыта.

**position\_by**

Тикет встречной позиции. Используется при закрытии позиции встречной — открытой по тому же инструменту, но в противоположном направлении.

При модификации или закрытии позиции в системе хеджинга обязательно указывайте ее тикет (`MqlTradeRequest::position`). В системе неттинга тикет также можно указывать, однако идентификации позиции осуществляется по имени символа.

Для отправки приказов на совершение [торговых операций](#) необходимо использовать функцию `OrderSend()`. Для каждой торговой операции необходимо указывать обязательные поля и можно заполнять optionalные поля. Всего предусмотрено семь вариантов отправки торгового запроса:

#### **Request Execution**

Торговый ордер на открытие позиции в режиме Request Execution (режим торговли по запросу текущих цен). Требуется указание 9 полей:

- action
- symbol
- volume
- price
- sl
- tp
- deviation
- type
- type\_filling

Можно также задать значения полей `magic` и `comment`.

#### **Instant Execution**

Торговый ордер на открытие позиции в режиме Instant Execution (режим торговли по потоковым ценам). Требуется указание 9 полей:

- action
- symbol
- volume
- price
- sl
- tp
- deviation
- type
- type\_filling

Можно также задать значения полей `magic` и `comment`.

#### **Market Execution**

Торговый ордер на открытие позиции в режиме Market Execution (режим исполнения торговых приказов по рынку). Требуется указание 5 полей:

- action

- symbol
- volume
- type
- type\_filling

Можно также задать значения полей magic и comment.

### Exchange Execution

Торговый ордер на открытие позиции в режиме Exchange Execution (биржевой режим исполнения торговых приказов). Требуется указание 5 полей:

- action
- symbol
- volume
- type
- type\_filling

Можно также задать значения полей magic и comment.

Пример торговой операции [TRADE\\_ACTION\\_DEAL](#) для открытия позиции Buy:

```
#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Открытие позиции Buy |
//+-----+
void OnStart()
{
//--- объявление и инициализация запроса и результата
MqlTradeRequest request={0};
MqlTradeResult result={0};
//--- параметры запроса
request.action =TRADE_ACTION_DEAL; // тип торговой операции
request.symbol =Symbol(); // символ
request.volume =0.1; // объем в 0.1 лот
request.type =ORDER_TYPE_BUY; // тип ордера
request.price =SymbolInfoDouble(Symbol(),SYMBOL_ASK); // цена для открытия
request.deviation=5; // допустимое отклонение
request.magic =EXPERT_MAGIC; // MagicNumber ордера
//--- отправка запроса
if(!OrderSend(request,result))
PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
//--- информация об операции
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
```

Пример торговой операции [TRADE\\_ACTION\\_DEAL](#) для открытия позиции Sell:

```

#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Открытие позиции Sell |
//+-----+
void OnStart()
{
//--- объявление и инициализация запроса и результата
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
//--- параметры запроса
    request.action =TRADE_ACTION_DEAL; // тип торговой операции
    request.symbol =Symbol(); // символ
    request.volume =0.2; // объем в 0.2 лот
    request.type =ORDER_TYPE_SELL; // тип ордера
    request.price =SymbolInfoDouble(Symbol(),SYMBOL_BID); // цена для открытия
    request.deviation=5; // допустимое отклонение
    request.magic =EXPERT_MAGIC; // MagicNumber ордера
//--- отправка запроса
    if(!OrderSend(request,result))
        PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
//--- информация об операции
    PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result
}

```

Пример торговой операции **TRADE\_ACTION\_DEAL** для закрытия позиций:

```

#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Закрытие всех позиций |
//+-----+
void OnStart()
{
//--- объявление запроса и результата
    MqlTradeRequest request;
    MqlTradeResult result;
    int total=PositionsTotal(); // количество открытых позиций
//--- перебор всех открытых позиций
    for(int i=total-1; i>=0; i--)
    {
//--- параметры ордера
        ulong position_ticket=PositionGetTicket(i);
        string position_symbol=PositionGetString(POSITION_SYMBOL);
        int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
        ulong magic=PositionGetInteger(POSITION_MAGIC);
        double volume=PositionGetDouble(POSITION_VOLUME);
        ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
//--- вывод информации о позиции
        PrintFormat("#%I64u %s %s %.2f %s [%I64d]",
                    position_ticket,
                    position_symbol,
                    EnumToString(type),
                    volume,
                    DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
                    magic);
//--- если MagicNumber совпадает
        if(magic==EXPERT_MAGIC)
        {
//--- обнуление значений запроса и результата
            ZeroMemory(request);
            ZeroMemory(result);
//--- установка параметров операции
            request.action =TRADE_ACTION_DEAL; // тип торговой операции
            request.position =position_ticket; // тикет позиции
            request.symbol =position_symbol; // символ
            request.volume =volume; // объем позиции
            request.deviation=5; // допустимое отклонение от цены
            request.magic =EXPERT_MAGIC; // MagicNumber позиции
//--- установка цены и типа ордера в зависимости от типа позиции
            if(type==POSITION_TYPE_BUY)
            {
                request.price=SymbolInfoDouble(position_symbol,SYMBOL_BID);
                request.type =ORDER_TYPE_SELL;
            }
            else
            {
                request.price=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
                request.type =ORDER_TYPE_BUY;
            }
//--- вывод информации о закрытии
            PrintFormat("Close %I64d %s %s",position_ticket,position_symbol,EnumToString(type));
//--- отправка запроса
            if(!OrderSend(request,result))
                PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
//--- информация об операции
            PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,
//---
        }
    }
}

```

```

    }
}

```

## SL & TP Modification

Торговый приказ на модификацию уровней StopLoss и/или TakeProfit. Требуется указание 4 полей:

- action
- symbol
- sl
- tp
- position

Пример торговой операции **TRADE\_ACTION\_SLTP** для изменения значений Stop Loss и Take Profit у открытой позиции:

```

#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Модификация Stop Loss и Take Profit позиции |
//+-----+
void OnStart()
{
//--- объявление запроса и результата
MqlTradeRequest request;
MqlTradeResult result;
int total=PositionsTotal(); // количество открытых позиций
//--- перебор всех открытых позиций
for(int i=0; i<total; i++)
{
//--- параметры ордера
ulong position_ticket=PositionGetTicket(i); // тикет позиции
string position_symbol=PositionGetString(POSITION_SYMBOL); // символ
int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS); // количество знаков после запятой
ulong magic=PositionGetInteger(POSITION_MAGIC); // MagicNumber позиции
double volume=PositionGetDouble(POSITION_VOLUME); // объем позиции
double sl=PositionGetDouble(POSITION_SL); // Stop Loss позиции
double tp=PositionGetDouble(POSITION_TP); // Take Profit позиции
ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
//--- вывод информации о позиции
PrintFormat("#%I64u %s %s %.2f %s sl: %s tp: %s [%I64d]",
position_ticket,
position_symbol,
EnumToString(type),
volume,
DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
DoubleToString(sl,digits),
DoubleToString(tp,digits),
magic);
//--- если MagicNumber совпадает, Stop Loss и Take Profit не заданы
if(magic==EXPERT_MAGIC && sl==0 && tp==0)
{
}
}

```

```

//--- вычисление текущих ценовых уровней
double price=PositionGetDouble(POSITION_PRICE_OPEN);
double bid=SymbolInfoDouble(position_symbol,SYMBOL_BID);
double ask=SymbolInfoDouble(position_symbol,SYMBOL_ASK);
int stop_level=(int)SymbolInfoInteger(position_symbol,SYMBOL_TRADE_STOPS_LEVEL);
double price_level;
//--- если уровень минимально допустимого отступа в пунктах от текущей цены <=0
if(stop_level<=0)
    stop_level=150; // зададим отступ в 150 пунктов от текущей цены закрытия
else
    stop_level+=50; // уровень отступа возьмем равным (SYMBOL_TRADE_STOPS_LEVEL)

//--- вычисление и округление значений Stop Loss и Take Profit
price_level=stop_level*SymbolInfoDouble(position_symbol,SYMBOL_POINT);
if(type==POSITION_TYPE_BUY)
{
    sl=NormalizeDouble(bid-price_level,digits);
    tp=NormalizeDouble(bid+price_level,digits);
}
else
{
    sl=NormalizeDouble(ask+price_level,digits);
    tp=NormalizeDouble(ask-price_level,digits);
}
//--- обнуление значений запроса и результата
ZeroMemory(request);
ZeroMemory(result);
//--- установка параметров операции
request.action =TRADE_ACTION_SLTP; // тип торговой операции
request.position=position_ticket; // тикет позиции
request.symbol=position_symbol; // символ
request.sl =sl; // Stop Loss позиции
request.tp =tp; // Take Profit позиции
request.magic=EXPERT_MAGIC; // MagicNumber позиции
//--- вывод информации о модификации
PrintFormat("Modify #%"I64d "s %s",position_ticket,position_symbol,EnumToString(request.action));
//--- отправка запроса
if(!OrderSend(request,result))
    PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
//--- информация об операции
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
}
}

```

## Pending Order

Торговый приказ на установку отложенного ордера. Требуется указание 11 полей:

- action
  - symbol
  - volume
  - price
  - stoplimit
  - sl
  - tp
  - type
  - type\_filling

- type\_time
- expiration

Можно также задать значения полей magic и comment.

Пример торговой операции **TRADE\_ACTION\_PENDING** для установки отложенного ордера:

```

#property description "Пример установки отложенных ордеров"
#property script_show_inputs
#define EXPERT_MAGIC 123456                                // MagicNumber эксперта
input ENUM_ORDER_TYPE orderType=ORDER_TYPE_BUY_LIMIT;    // тип ордера
//+-----+
//| Установка отложенных ордеров                         |
//+-----+
void OnStart()
{
//-- объявление и инициализация запроса и результата
MqlTradeRequest request={0};
MqlTradeResult result={0};
//-- параметры для установки отложенного ордера
request.action =TRADE_ACTION_PENDING;                   // тип торговой
request.symbol =Symbol();                             // символ
request.volume =0.1;                                 // объем в 0.1
request.deviation=2;                                // допустимое с
request.magic =EXPERT_MAGIC;                         // MagicNumber
int offset = 50;                                    // отступ от т
double price;                                       // цена срабатыв
double point=SymbolInfoDouble(_Symbol,SYMBOL_POINT); // размер пункта
int digits=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS); // кол-во знаков
//-- проверка типа операции
if(orderType==ORDER_TYPE_BUY_LIMIT)
{
    request.type =ORDER_TYPE_BUY_LIMIT;                // тип ордера
    price=SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point; // цена для от
    request.price =NormalizeDouble(price,digits);       // нормализован
}
else if(orderType==ORDER_TYPE_SELL_LIMIT)
{
    request.type =ORDER_TYPE_SELL_LIMIT;               // тип ордера
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point; // цена для от
    request.price =NormalizeDouble(price,digits);       // нормализован
}
else if(orderType==ORDER_TYPE_BUY_STOP)
{
    request.type =ORDER_TYPE_BUY_STOP;                 // тип ордера
    price =SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point; // цена для от
    request.price=NormalizeDouble(price,digits);        // нормализован
}
else if(orderType==ORDER_TYPE_SELL_STOP)
{
    request.type =ORDER_TYPE_SELL_STOP;                // тип ордера
    price=SymbolInfoDouble(Symbol(),SYMBOL_BID)-offset*point; // цена для от
    request.price =NormalizeDouble(price,digits);       // нормализован
}
else Alert("Этот пример только для установки отложенных ордеров"); // если выбран
//-- отправка запроса
if(!OrderSend(request,result))
    PrintFormat("OrderSend error %d",GetLastError()); // если отправка
//-- информация об операции
PrintFormat("retcode=%I64u deal=%I64u order=%I64u",result.retcode,result.deal,result
}
//+-----+

```

### Modify Pending Order

Торговый приказ на модификацию уровней цен отложенного ордера. Требуется указание 7 полей:

- action
- order
- price
- sl
- tp
- type\_time
- expiration

Пример торговой операции [TRADE\\_ACTION MODIFY](#) для модификации уровней цен отложенного ордера:

```

#define EXPERT_MAGIC 123456 // MagicNumber эксперта
//+-----+
//| Модификация отложенных ордеров |
//+-----+
void OnStart()
{
//-- объявление и инициализация запроса и результата
    MqlTradeRequest request={0};
    MqlTradeResult result={0};
    int total=OrdersTotal(); // количество установленных отложенных ордеров
//--- перебор всех установленных отложенных ордеров
    for(int i=0; i<total; i++)
    {
//--- параметры ордера
        ulong order_ticket=OrderGetTicket(i); // тикет ордера
        string order_symbol=Symbol(); // символ
        int digits=(int)SymbolInfoInteger(order_symbol,SYMBOL_DIGITS); // количество знаков
        ulong magic=OrderGetInteger(ORDER_MAGIC); // MagicNumber
        double volume=OrderGetDouble(ORDER_VOLUME_CURRENT); // текущий объем
        double sl=OrderGetDouble(ORDER_SL); // текущий Stop Loss
        double tp=OrderGetDouble(ORDER_TP); // текущий Take Profit
        ENUM_ORDER_TYPE type=(ENUM_ORDER_TYPE)OrderGetInteger(ORDER_TYPE); // тип ордера
        int offset = 50; // отступ от цены
        double price; // цена срабатывания
        double point=SymbolInfoDouble(order_symbol,SYMBOL_POINT); // размер пункта
//--- вывод информации об ордере
        PrintFormat("#%I64u %s %s %.2f %s sl: %s tp: %s [%I64d]", // формат вывода
            order_ticket,
            order_symbol,
            EnumToString(type),
            volume,
            DoubleToString(PositionGetDouble(POSITION_PRICE_OPEN),digits),
            DoubleToString(sl,digits),
            DoubleToString(tp,digits),
            magic);
//--- если MagicNumber совпадает, Stop Loss и Take Profit не заданы
        if(magic==EXPERT_MAGIC && sl==0 && tp==0)
        {
            request.action=TRADE_ACTION MODIFY; // тип торговой операции
            request.order = OrderGetTicket(i); // тикет ордера
            request.symbol =Symbol(); // символ
            request.deviation=5; // допустимое смещение
//--- установка уровня цены, тейк-профит и стоп-лосс ордера в зависимости от типа
            if(type==ORDER_TYPE_BUY_LIMIT)
            {
                price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)-offset*point;
                request.tp = NormalizeDouble(price+offset*point,digits);
                request.sl = NormalizeDouble(price-offset*point,digits);
                request.price =NormalizeDouble(price,digits); // нормализация
            }
            else if(type==ORDER_TYPE_SELL_LIMIT)
            {
                price = SymbolInfoDouble(Symbol(),SYMBOL_BID)+offset*point;
                request.tp = NormalizeDouble(price-offset*point,digits);
                request.sl = NormalizeDouble(price+offset*point,digits);
                request.price =NormalizeDouble(price,digits); // нормализация
            }
            else if(type==ORDER_TYPE_BUY_STOP)
            {
                price = SymbolInfoDouble(Symbol(),SYMBOL_ASK)+offset*point;
                request.tp = NormalizeDouble(price+offset*point,digits);
            }
        }
    }
}

```

```

        request.sl = NormalizeDouble(price-offset*point,digits);
        request.price    =NormalizeDouble(price,digits);                                // норма
    }
    else if(type==ORDER_TYPE_SELL_STOP)
    {
        price = SymbolInfoDouble(Symbol(),SYMBOL_BID)-offset*point;
        request.tp = NormalizeDouble(price-offset*point,digits);
        request.sl = NormalizeDouble(price+offset*point,digits);
        request.price    =NormalizeDouble(price,digits);                                // норма
    }
    //--- отправка запроса
    if(!OrderSend(request,result))
        PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
    //--- информация об операции
    PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,
    //--- обнуление значений запроса и результата
    ZeroMemory(request);
    ZeroMemory(result);
}
}
//+-----+

```

### Delete Pending Order

Торговый приказ на удаление отложенного ордера. Требуется указание 2 полей:

- action
- order

Пример торговой операции **TRADE\_ACTION\_REMOVE** для удаления отложенных ордеров:

```
#define EXPERT_MAGIC 123456 // MagicNumber эксперта
```

```

//+-----+
//| Удаление отложенных ордеров
//+-----+
void OnStart()
{
//-- объявление и инициализация запроса и результата
MqlTradeRequest request={0};
MqlTradeResult result={0};
int total=OrdersTotal(); // количество установленных отложенных ордеров
//--- перебор всех установленных отложенных ордеров
for(int i=total-1; i>=0; i--)
{
    ulong order_ticket=OrderGetTicket(i);                                // тикет ордера
    ulong magic=OrderGetInteger(ORDER_MAGIC);                            // MagicNumber ордера
    //--- если MagicNumber совпадает
    if(magic==EXPERT_MAGIC)
    {
        //--- обнуление значений запроса и результата
        ZeroMemory(&request);
        ZeroMemory(&result);
        //--- установка параметров операции
        request.action=TRADE_ACTION_REMOVE;                                // тип торговой операции
        request.order = order_ticket;                                       // тикет ордера
        //--- отправка запроса
        if(!OrderSend(request,result))
            PrintFormat("OrderSend error %d",GetLastError()); // если отправить запрос
        //--- информация об операции
        PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,
    }
}
//+-----+

```

#### Смотри также

[Структуры и классы](#), [Торговые функции](#), [Свойства ордеров](#)

## Структура результата проверки торгового запроса (MqlTradeCheckResult)

Прежде чем [отправить](#) торговому серверу [запрос](#) на [торговую операцию](#), рекомендуется провести его проверку. Проверка осуществляется функцией [OrderCheck\(\)](#), которой передается сам проверяемый запрос и переменная типа структуры MqlTradeCheckResult. В эту переменную и будет записан результат проверки.

```
struct MqlTradeCheckResult
{
    uint      retcode;           // Код ответа
    double    balance;          // Баланс после совершения сделки
    double    equity;           // Эквити после совершения сделки
    double    profit;           // Плавающая прибыль
    double    margin;           // Маржевые требования
    double    margin_free;      // Свободная маржа
    double    margin_level;     // Уровень маржи
    string   comment;          // Комментарий к коду ответа (описание ошибки)
};
```

### Описание полей

Поле	Описание
retcode	<a href="#">Код возврата</a>
balance	Значение баланса, которое будет после выполнения торговой операции
equity	Значение собственных средств, которое будет после выполнения торговой операции
profit	Значение плавающей прибыли, которое будет после выполнения торговой операции
margin	Размер маржи необходимый для требуемой торговой операции
margin_free	Размер свободных собственных средств, которые останутся после выполнения требуемой торговой операции
margin_level	Уровень маржи, который установится после выполнения требуемой торговой операции
comment	Комментарий к коду ответа, описание ошибки

### Смотри также

[Структура торгового запроса](#), [Структура для получения текущих цен](#), [OrderSend](#), [OrderCheck](#)

## Структура результата торгового запроса (MqlTradeResult)

В ответ на [торговый запрос](#) постановки ордера в торговую систему, торговый сервер возвращает данные, содержащие информацию о результате обработки торгового запроса в виде специальной предопределенной структуры `MqlTradeResult`.

```
struct MqlTradeResult
{
    uint     retcode;           // Код результата операции
    ulong    deal;             // Тикет сделки, если она совершена
    ulong    order;            // Тикет ордера, если он выставлен
    double   volume;           // Объем сделки, подтверждённый брокером
    double   price;            // Цена в сделке, подтверждённая брокером
    double   bid;              // Текущая рыночная цена предложения (цены реквоты)
    double   ask;              // Текущая рыночная цена спроса (цены реквоты)
    string   comment;          // Комментарий брокера к операции (по умолчанию заполняется пустой строкой)
    uint     request_id;        // Идентификатор запроса, устанавливается терминалом при отправке запроса
    uint     retcode_external; // Код ответа внешней торговой системы
};
```

### Описание полей

Поле	Описание
retcode	<a href="#">Код возврата</a> торгового сервера
deal	Тикет <a href="#">сделки</a> , если она совершена. Сообщается при торговой операции <a href="#">TRADE_ACTION_DEAL</a>
order	Тикет <a href="#">ордера</a> , если он выставлен. Сообщается при торговой операции <a href="#">TRADE_ACTION_PENDING</a>
volume	Объем сделки, подтверждённый брокером. Зависит от <a href="#">типа ордера по исполнению</a>
price	Цена в сделке, подтверждённая брокером. Зависит от поля <i>deviation</i> в <a href="#">торговом запросе</a> и/или от типа <a href="#">торговой операции</a>
bid	Текущая рыночная цена предложения (цены реквоты)
ask	Текущая рыночная цена спроса (цены реквоты)
comment	Комментарий брокера к операции (по умолчанию заполняется расшифровкой кода возврата торгового сервера)
request_id	Идентификатор запроса, проставляемый терминалом при отсылке на торговый сервер

## retcode\_external

Код ошибки, которую вернула внешняя торговая система. Проставление и виды этих ошибок зависят от брокера и внешней торговой системы, в которую выводятся торговые операции

Результат торговой операции возвращается в переменную типа `MqlTradeResult`, которая передается вторым параметром в функцию [OrderSend\(\)](#) для проведения [торговых операций](#).

Терминал записывает идентификатор [запроса](#) в поле `request_id` при его отправке на торговый сервер функциями [OrdersSend\(\)](#) и [OrderSendAsync\(\)](#). От торгового сервера терминал получает сообщения о совершенных торговых транзакциях и передает их на обработку в функцию [OnTradeTransaction\(\)](#), которая содержит в качестве параметров:

- описание самой торговой транзакции в структуре [MqlTradeTransaction](#);
- описание [торгового запроса](#), отправленного из функции `OrderSend()` или `OrdersSendAsync()`. Идентификатор запроса отправляется терминалом на торговый сервер, а сам запрос и его `request_id` сохраняются в памяти терминала;
- результат исполнения торгового запроса в виде структуры `MqlTradeResult`, в котором поле `request_id` содержит идентификатор этого самого запроса.

Функция `OnTradeTransaction()` получает три входных параметра, но последние два параметра имеет смысл анализировать только для торговых транзакций, имеющих тип [TRADE TRANSACTION REQUEST](#). Во всех остальных случаях данные о торговом запросе и результате его выполнения не заполняются. Пример анализа параметров приведен в разделе [Структура торговой транзакции](#).

Установка терминалом идентификатора `request_id` для торгового запроса при его отправке на сервер в первую очередь предназначена для работы с асинхронной функцией `OrderSendAsync()`. Этот идентификатор позволяет связать выполненное действие (вызов функций `OrderSend` или `OrderSendAsync`) с результатом этого действия, передаваемым в [OnTradeTransaction\(\)](#).

### Пример:

```
//+-----+
//| Отправка торгового запроса с обработкой результата |
//+-----+
bool MyOrderSend (MqlTradeRequest request,MqlTradeResult result)
{
//--- сбросим код последней ошибки в ноль
    ResetLastError();
//--- отправим запрос
    bool success=OrderSend(request,result);
//--- если результат неудачный - попробуем узнать в чем дело
    if(!success)
    {
        int answer=result.retcode;
        Print("TradeLog: Trade request failed. Error = ",GetLastError());
        switch(answer)
        {
//--- реквота
        }
    }
}
```

```

case 10004:
{
    Print("TRADE_RETCODE_QUOTE");
    Print("request.price = ",request.price," result.ask = ",
          result.ask," result.bid = ",result.bid);
    break;
}
//--- ордер не принят сервером
case 10006:
{
    Print("TRADE_RETCODE_REJECT");
    Print("request.price = ",request.price," result.ask = ",
          result.ask," result.bid = ",result.bid);
    break;
}
//--- неправильная цена
case 10015:
{
    Print("TRADE_RETCODE_INVALID_PRICE");
    Print("request.price = ",request.price," result.ask = ",
          result.ask," result.bid = ",result.bid);
    break;
}
//--- неправильный SL и/или ТР
case 10016:
{
    Print("TRADE_RETCODE_INVALID_STOPS");
    Print("request.sl = ",request.sl," request.tp = ",request.tp);
    Print("result.ask = ",result.ask," result.bid = ",result.bid);
    break;
}
//--- некорректный объем
case 10014:
{
    Print("TRADE_RETCODE_INVALID_VOLUME");
    Print("request.volume = ",request.volume," result.volume = ",
          result.volume);
    break;
}
//--- не хватает денег на торговую операцию
case 10019:
{
    Print("TRADE_RETCODE_NO_MONEY");
    Print("request.volume = ",request.volume," result.volume = ",
          result.volume," result.comment = ",result.comment);
    break;
}
//--- какая-то другая причина, сообщим код ответа сервера
default:

```

```
{  
    Print("Other answer = ",answer);  
}  
}  
//--- сообщим о неудачном результате торгового запроса возвратом false  
return(false);  
}  
//--- OrderSend() вернул true - повторим ответ  
return(true);  
}
```

## Структура торговой транзакции (MqlTradeTransaction)

В результате выполнения определенных действий с торговым счетом, его состояние изменяется. К таким действиям относятся:

- Отсылка торгового запроса любым MQL5-приложением в клиентском терминале при помощи функций [OrderSend](#) и [OrderSendAsync](#) и его последующее исполнение;
- Отсылка торгового запроса через графический интерфейс терминала и его последующее исполнение;
- Срабатывания отложенных ордеров и стоп-ордеров на сервере;
- Выполнения операций на стороне торгового сервера.

В результате данных действий для счета выполняются торговые транзакции:

- обработка торгового запроса;
- изменение открытых ордеров;
- изменение истории ордеров;
- изменение истории сделок;
- изменение позиций.

Например, при отсылке рыночного ордера на покупку он обрабатывается, для счета создается соответствующий ордер на покупку, происходит исполнение ордера, его удаление из списка открытых, добавление в историю ордеров, далее добавляется соответствующая сделка в историю и создается новая позиция. Все эти действия являются [торговыми транзакциями](#).

Для получения торговых транзакций, примененных к счету, в MQL5 предусмотрен специальный обработчик [OnTradeTransaction\(\)](#). В первый параметр этого обработчика передается структура MqlTradeTransaction, описывающая торговые транзакции.

```
struct MqlTradeTransaction
{
    ulong deal; // Тикет сделки
    ulong order; // Тикет ордера
    string symbol; // Имя торгового инструмента
    ENUM_TRADE_TRANSACTION_TYPE type; // Тип торговой транзакции
    ENUM_ORDER_TYPE order_type; // Тип ордера
    ENUM_ORDER_STATE order_state; // Состояние ордера
    ENUM_DEAL_TYPE deal_type; // Тип сделки
    ENUM_ORDER_TYPE_TIME time_type; // Тип ордера по времени действия
    datetime time_expiration; // Срок истечения ордера
    double price; // Цена
    double price_trigger; // Цена срабатывания стоп-лимитного ордера
    double price_sl; // Уровень Stop Loss
    double price_tp; // Уровень Take Profit
    double volume; // Объем в лотах
    ulong position; // Тикет позиции
    ulong position_by; // Тикет встречной позиции
};
```

## Описание полей

Поле	Описание
deal	Тикет сделки.
order	Тикет ордера.
symbol	Имя торгового инструмента, по которому совершена транзакция.
type	Тип торговой транзакции. Значение может быть одним из значений перечисления <a href="#">ENUM_TRADE_TRANSACTION_TYPE</a> .
order_type	Тип торгового ордера. Значение может быть одним из значений перечисления <a href="#">ENUM_ORDER_TYPE</a> .
order_state	Состояние торгового ордера. Значение может быть одним из значений перечисления <a href="#">ENUM_ORDER_STATE</a> .
deal_type	Тип сделки. Значение может быть одним из значений перечисления <a href="#">ENUM DEAL TYPE</a> .
type_time	Тип ордера по истечению. Значение может быть одним из значений перечисления <a href="#">ENUM_ORDER_TYPE_TIME</a> .
time_expiration	Срок истечения отложенного ордера (для ордеров типа <a href="#">ORDER_TIME_SPECIFIED</a> и <a href="#">ORDER_TIME_SPECIFIED_DAY</a> ).
price	Цена. В зависимости от типа торговой транзакции может быть ценой ордера, сделки или позиции.
price_trigger	Стоп-цена (цена срабатывания) стоп-лимитного ордера ( <a href="#">ORDER_TYPE_BUY_STOP_LIMIT</a> и <a href="#">ORDER_TYPE_SELL_STOP_LIMIT</a> ).
price_sl	Цена Stop Loss. В зависимости от типа торговой транзакции может относиться к ордеру, сделке или позиции.
price_tp	Цена Take Profit. В зависимости от типа торговой транзакции может относиться к ордеру, сделке или позиции.
volume	Объем в лотах. В зависимости от типа торговой транзакции может указывать на текущий объем ордера, объем сделки или объем позиции.

position	Тикет позиции, на которую повлияла транзакция.
position_by	Тикет встречной позиции. Используется при закрытии позиции встречной — открытой по тому же инструменту, но в противоположном направлении.

Определяющим параметром для анализа поступившей транзакции является ее тип, который передается в поле `type`. Например, если транзакция является типом `TRADE_TRANSACTION_REQUEST` (получен результат обработки торгового запроса сервером), то структура имеет только одно заполненное поле `type`, остальные поля анализировать не нужно. В этом случае можно произвести анализ двух дополнительных параметров `request` и `result`, которые передаются в обработчик `OnTradeTransaction()`, как это показано в примере ниже.

Зная тип торговой операции, можно принять решение об анализе текущего состояния ордеров, позиций и сделок на торговом счете. Необходимо иметь в виду, что один торговый запрос, отправленный из терминала серверу, может породить несколько торговых транзакций, очередность поступления которых в терминал не гарантировается.

Структура `MqlTradeTransaction` заполняется по-разному в зависимости от типа торговой транзакции ([ENUM\\_TRADE\\_TRANSACTION\\_TYPE](#)):

#### TRADE\_TRANSACTION\_ORDER\_\* и TRADE\_TRANSACTION\_HISTORY\_\*

Для торговых транзакций, касающихся обработки открытых ордеров (`TRADE_TRANSACTION_ORDER_ADD`, `TRADE_TRANSACTION_ORDER_UPDATE` и `TRADE_TRANSACTION_ORDER_DELETE`) и истории ордеров (`TRADE_TRANSACTION_HISTORY_ADD`, `TRADE_TRANSACTION_HISTORY_UPDATE`, `TRADE_TRANSACTION_HISTORY_DELETE`), в структуре `MqlTradeTransaction` заполняются следующие поля:

- `order` - тикет ордера;
- `symbol` - имя финансового инструмента в ордере;
- `type` - тип торговой транзакции;
- `order_type` - тип ордера;
- `orders_state` - текущее состояние ордера;
- `time_type` - тип истечения ордера;
- `time_expiration` - время истечения ордера (для ордеров с типом истечения [ORDER\\_TIME\\_SPECIFIED](#) и [ORDER\\_TIME\\_SPECIFIED\\_DAY](#));
- `price` - цена в ордере, указанная клиентом;
- `price_trigger` - стоп-цена срабатывания стоп-лимитного ордера (только для [ORDER\\_TYPE\\_BUY\\_STOP\\_LIMIT](#) и [ORDER\\_TYPE\\_SELL\\_STOP\\_LIMIT](#));
- `price_sl` - цена Stop Loss ордера (заполняется, если указана в ордере);
- `price_tp` - цена Take Profit ордера (заполняется, если указана в ордере);
- `volume` - текущий объем ордера (не исполненный). Изначальный объем ордера можно узнать из истории ордеров при помощи функций [HistoryOrders](#)\*
- `position` - тикет позиции, открытой, измененной или закрытой в результате исполнения ордера. Заполняется только для рыночных ордеров. Не заполняется для `TRADE_TRANSACTION_ORDER_ADD`.
- `position_by` - тикет встречной позиции. Заполняется только для ордеров на закрытие позиции встречной (`close by`).

### TRADE\_TRANSACTION\_DEAL\_\*

Для торговых транзакций, касающихся обработки сделок (TRADE\_TRANSACTION DEAL\_ADD, TRADE\_TRANSACTION DEAL\_UPDATE и TRADE\_TRANSACTION DEAL\_DELETE), в структуре MqlTradeTransaction заполняются следующие поля:

- deal - тикет сделки;
- order - тикет ордера, на основе которого совершена сделка;
- symbol - имя финансового инструмента в сделке;
- type - тип торговой транзакции;
- deal\_type - тип сделки;
- price - цена, по которой совершена сделка;
- price\_sl - цена Stop Loss (заполняется, если указана в ордере, на основе которого совершена сделка);
- price\_tp - цена Take Profit (заполняется, если указана в ордере, на основе которого совершена сделка);
- volume - объем сделки в лотах.
- position - тикет позиции, открытой, измененной или закрытой в результате исполнения сделки.
- position\_by - тикет встречной позиции. Заполняется только для сделок на закрытие позиции встречной (out by).

### TRADE\_TRANSACTION\_POSITION

Для торговых транзакций, касающихся изменений позиций, не связанных с исполнением сделок (TRADE\_TRANSACTION POSITION), в структуре MqlTradeTransaction заполняются следующие поля:

- symbol - имя финансового инструмента позиции;
- type - тип торговой транзакции;
- deal\_type - тип позиции ([DEAL\\_TYPE\\_BUY](#) или [DEAL\\_TYPE\\_SELL](#));
- price - средневзвешенная цена открытия позиции;
- price\_sl - цена Stop Loss;
- price\_tp - цена Take Profit;
- volume - объем позиции в лотах, если он был изменен.
- position - тикет позиции.

Изменение позиции (добавление, изменение или ликвидация) в результате совершения сделки не влечет за собой появление транзакции TRADE\_TRANSACTION\_POSITION.

### TRADE\_TRANSACTION\_REQUEST

Для торговых транзакций, описывающих факт, что торговый запрос обработан сервером, и результат его обработки получен (TRADE\_TRANSACTION REQUEST), в структуре MqlTradeTransaction заполняется только одно поле:

- type - тип торговой транзакции;

Для транзакций данного типа необходимо анализировать только одно поле - type (тип торговой транзакции). Для получения дополнительной информации необходимо анализировать второй и третий параметры функции [OnTradeTransaction](#) (request и result).

**Пример:**

```

input int MagicNumber=1234567;

//--- подключим торговый класс CTrade и объявим переменную этого типа
#include <Trade\Trade.mqh>
CTrade trade;
//--- флаги для установки и удаления отложенного ордера
bool pending_done=false;
bool pending_deleted=false;
//--- здесь будем хранить тикет отложенного ордера
ulong order_ticket;
//+-----+
//| Expert initialization function | |
//+-----+
int OnInit()
{
//--- установим MagicNumber, которым будут помечаться все наши ордера
trade.SetExpertMagicNumber(MagicNumber);
//--- торговые запросы будем отправлять в асинхронном режиме с помощью функции OrderSend()
trade.SetAsyncMode(true);
//--- инициализируем переменную нулем
order_ticket=0;
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Expert tick function | |
//+-----+
void OnTick()
{
//--- установка отложенного ордера
if(!pending_done)
{
double ask=SymbolInfoDouble(_Symbol,SYMBOL_ASK);
double buy_stop_price=NormalizeDouble(ask+1000*_Point,(int)SymbolInfoInteger(_Symbol,SYMBOL_DIGITS));
bool res=trade.BuyStop(0.1,buy_stop_price,_Symbol);
//--- если функция BuyStop() отработала успешно
if(res)
{
pending_done=true;
//--- получим результат отправки запроса из ctrade
MqlTradeResult trade_result;
trade.Result(trade_result);
//--- получим request_id для отправленного запроса
uint request_id=trade_result.request_id;
Print("Отправлен запрос на установку отложенного ордера. Идентификатор запроса: " + IntToString(request_id));
//--- запомним тикет ордера (при использовании асинхронного режима отправки запросов)
order_ticket=trade_result.order;
//--- всё сделано, поэтому досрочно выходим из обработчика OnTick()
return;
}
}
}

```

```

        }

    }

//--- удаление отложенного ордера
if(!pending_deleted)
{
    //--- дополнительная проверка
    if(pending_done && (order_ticket!=0))
    {
        //--- попытаемся удалить отложенный ордер
        bool res=trade.OrderDelete(order_ticket);
        Print("OrderDelete=",res);
        //--- при успешной отправке запроса на удаление
        if(res)
        {
            pending_deleted=true;
            //--- получим результат выполнения запроса
            MqlTradeResult trade_result;
            trade.Result(trade_result);
            //--- вытащим из результата идентификатор запроса
            uint request_id=trade_result.request_id;
            //--- выведем в Журнал
            Print("Отправлен запрос на удаление отложенного ордера #",order_ticket,
                  ". Идентификатор запроса Request_ID=",request_id,
                  "\r\n");
            //--- запишем из результата запроса тикет ордера
            order_ticket=trade_result.order;
        }
    }
}

//---

//+-----+
//| TradeTransaction function
//+-----+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
{
    //--- получим тип транзакции в виде значения перечисления
    ENUM_TRADE_TRANSACTION_TYPE type=(ENUM_TRADE_TRANSACTION_TYPE)trans.type;
    //--- если транзакция является результатом обработки запроса, выведем только её название
    if(type==TRADE_TRANSACTION_REQUEST)
    {
        Print(EnumToString(type));
        //--- выведем строковое описание обработанного запроса
        Print("-----RequestDescription\r\n",RequestDescription(request));
        //--- выведем описание результата запроса
        Print("-----ResultDescription\r\n",TradeResultDescription(result));
        //--- запомним тикет ордера для его удаления на следующей обработке в OnTick()
        if(result.order!=0)
        {
    }
}

```

```

//--- удалим этот ордер по его тикету при следующем вызове OnTick()
order_ticket=result.order;
Print(" Тикет отложенного ордера ",order_ticket,"\r\n");
}

}

else // для транзакций другого типа выведем полное описание
//--- выведем описание полученной транзакции в Журнал
Print("-----TransactionDescription\r\n",TransactionDescription(trans));

//---

}

//+-----+
//| Возвращает текстовое описание транзакции |
//+-----+

string TransactionDescription(const MqlTradeTransaction &trans)
{
//---

    string desc=EnumToString(trans.type)+"\r\n";
    desc+="Symbol: "+trans.symbol+"\r\n";
    desc+="Deal ticket: "+(string)trans.deal+"\r\n";
    desc+="Deal type: "+EnumToString(trans.deal_type)+"\r\n";
    desc+="Order ticket: "+(string)trans.order+"\r\n";
    desc+="Order type: "+EnumToString(trans.order_type)+"\r\n";
    desc+="Order state: "+EnumToString(trans.order_state)+"\r\n";
    desc+="Order time type: "+EnumToString(trans.time_type)+"\r\n";
    desc+="Order expiration: "+TimeToString(trans.time_expiration)+"\r\n";
    desc+="Price: "+StringFormat("%G",trans.price)+"\r\n";
    desc+="Price trigger: "+StringFormat("%G",trans.price_trigger)+"\r\n";
    desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"\r\n";
    desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"\r\n";
    desc+="Volume: "+StringFormat("%G",trans.volume)+"\r\n";
    desc+="Position: "+(string)trans.position+"\r\n";
    desc+="Position by: "+(string)trans.position_by+"\r\n";

//--- вернем полученную строку
    return desc;
}

//+-----+
//| Возвращает текстовое описание торгового запроса |
//+-----+

string RequestDescription(const MqlTradeRequest &request)
{
//---

    string desc=EnumToString(request.action)+"\r\n";
    desc+="Symbol: "+request.symbol+"\r\n";
    desc+="Magic Number: "+StringFormat("%d",request.magic)+"\r\n";
    desc+="Order ticket: "+(string)request.order+"\r\n";
    desc+="Order type: "+EnumToString(request.type)+"\r\n";
    desc+="Order filling: "+EnumToString(request.type_filling)+"\r\n";
    desc+="Order time type: "+EnumToString(request.type_time)+"\r\n";
}

```

```

desc+="Order expiration: "+TimeToString(request.expiration)+"\r\n";
desc+="Price: "+StringFormat("%G",request.price)+"\r\n";
desc+="Deviation points: "+StringFormat("%G",request.deviation)+"\r\n";
desc+="Stop Loss: "+StringFormat("%G",request.sl)+"\r\n";
desc+="Take Profit: "+StringFormat("%G",request.tp)+"\r\n";
desc+="Stop Limit: "+StringFormat("%G",request.stoplimit)+"\r\n";
desc+="Volume: "+StringFormat("%G",request.volume)+"\r\n";
desc+="Comment: "+request.comment+"\r\n";
//--- вернем полученную строку
return desc;
}

//+-----+
//| Возвращает текстовое описание результата обработки запроса |+
//+-----+
string TradeResultDescription(const MqlTradeResult &result)
{
//---

    string desc="Retcode "+(string)result.retcode+"\r\n";
    desc+="Request ID: "+StringFormat("%d",result.request_id)+"\r\n";
    desc+="Order ticket: "+(string)result.order+"\r\n";
    desc+="Deal ticket: "+(string)result.deal+"\r\n";
    desc+="Volume: "+StringFormat("%G",result.volume)+"\r\n";
    desc+="Price: "+StringFormat("%G",result.price)+"\r\n";
    desc+="Ask: "+StringFormat("%G",result.ask)+"\r\n";
    desc+="Bid: "+StringFormat("%G",result.bid)+"\r\n";
    desc+="Comment: "+result.comment+"\r\n";
//--- вернем полученную строку
return desc;
}

```

#### Смотри также

[Типы торговых транзакций](#), [OnTradeTransaction\(\)](#)

## Структура для получения текущих цен (MqlTick)

Структура для хранения последних цен по символу. Предназначена для быстрого получения наиболее востребованной информации о текущих ценах.

```
struct MqlTick
{
    datetime   time;           // Время последнего обновления цен
    double     bid;            // Текущая цена Bid
    double     ask;            // Текущая цена Ask
    double     last;           // Текущая цена последней сделки (Last)
    ulong      volume;         // Объем для текущей цены Last
    long       time_msc;       // Время последнего обновления цен в миллисекундах
    uint       flags;          // Флаги тиков
    double     volume_real;    // Объем для текущей цены Last с повышенной точностью
};
```

Переменная типа `MqlTick` позволяет за один вызов функции [SymbolInfoTick\(\)](#) получить значения Ask, Bid, Last и Volume.

У каждого тика всегда заполняются все параметры, независимо от того, изменились ли данные по сравнению с предыдущим тиком. Это позволяет всегда иметь актуальное состояние цен на любой момент времени без поиска предыдущих значений по тиковой истории. Например, с тиком могла измениться только цена бид, но в структуре помимо новой цены будут указаны и остальные параметры: предыдущая цена аск, объем и т.д.

Чтобы узнать, какие именно данные изменились с текущим тиком, анализируйте его флаги:

- `TICK_FLAG_BID` - тик изменил цену бид
- `TICK_FLAG_ASK` - тик изменил цену аск
- `TICK_FLAG_LAST` - тик изменил цену последней сделки
- `TICK_FLAG_VOLUME` - тик изменил объем
- `TICK_FLAG_BUY` - тик возник в результате сделки на покупку
- `TICK_FLAG_SELL` - тик возник в результате сделки на продажу

**Пример:**

```
void OnTick()
{
    MqlTick last_tick;
    //---
    if(SymbolInfoTick(Symbol(),last_tick))
    {
        Print(last_tick.time,": Bid = ",last_tick.bid,
              " Ask = ",last_tick.ask," Volume = ",last_tick.volume);
    }
    else
        Print("SymbolInfoTick() failed, error = ",GetLastError());
    //---
}
```

Смотри также

[Структуры и классы](#), [CopyTicks\(\)](#), [SymbolInfoTick\(\)](#)

## Структуры экономического календаря

В этом разделе описываются структуры для работы с [Экономическим календарем](#), который доступен прямо в платформе MetaTrader. Экономический календарь является готовой энциклопедией с описанием макроэкономических индикаторов, даты их выхода и степени важности. Актуальные значения макроэкономических показателей поступают в платформу MetaTrader моментально прямо в момент публикации и отображаются на графике в виде меток - это позволяет визуально отслеживать нужные показатели в разрезе стран, валют и важности.

[Функции Экономического календаря](#) позволяют проводить автоматический анализ поступающих событий по собственным критериям важности и в разрезе нужных стран/валютных пар.

Описания стран задаются структурой `MqlCalendarCountry`. Используется в функциях [CalendarCountryById\(\)](#) и [CalendarCountries\(\)](#)

```
struct MqlCalendarCountry
{
    ulong id; // идентификатор страны
    string name; // текстовое имя страны
    string code; // кодовое имя страны ISO
    string currency; // код валюты страны
    string currency_symbol; // символ/знак валюты страны
    string url_name; // имя страны, используемое в URL
};
```

Описания событий задаются структурой `MqlCalendarEvent`. Используется в функциях [CalendarEventById\(\)](#), [CalendarEventByCountry\(\)](#) и [CalendarEventByCurrency\(\)](#)

```
struct MqlCalendarEvent
{
    ulong id; // идентификатор события
    ENUM CALENDAR EVENT TYPE type; // тип события из перечисления
    ENUM CALENDAR EVENT SECTOR sector; // сектор, к которому относится событие
    ENUM CALENDAR EVENT FREQUENCY frequency; // частота (периодичность)
    ENUM CALENDAR EVENT TIMEMODE time_mode; // режим времени события
    ulong country_id; // идентификатор страны
    ENUM CALENDAR EVENT UNIT unit; // единица измерения значений
    ENUM CALENDAR EVENT IMPORTANCE importance; // важность события
    ENUM CALENDAR EVENT MULTIPLIER multiplier; // множитель значения этого события
    uint digits; // количество знаков после запятой
    string source_url; // URL источника, где публикуются данные
    string event_code; // код события
    string name; // текстовое имя события
};
```

Значения событий задаются структурой `MqlCalendarValue`. Используется в функциях `CalendarValueById()`, `CalendarValueHistoryByEvent()`, `CalendarValueHistory()`, `CalendarValueLastByEvent()` и `CalendarValueLast()`

```
struct MqlCalendarValue
{
    ulong id; // ID значения
    ulong event_id; // ID события
    datetime time; // время и дата события
    datetime period; // отчетный период события
    int revision; // ревизия публикуемого
    long actual_value; // актуальное значение
    long prev_value; // предыдущее значение
    long revised_prev_value; // пересмотренное предыдущее
    long forecast_value; // прогнозное значение
    ENUM CALENDAR_EVENT_IMPACT impact_type; // потенциальное влияние
};
```

Частота (периодичность) события указывается в структуре `MqlCalendarEvent`. Возможные значения указаны в перечислении `ENUM_CALENDAR_EVENT_FREQUENCY`

Идентификатор	Описание
CALENDAR_FREQUENCY_NONE	Частота публикации не задана
CALENDAR_FREQUENCY_WEEK	Публикация один раз в неделю
CALENDAR_FREQUENCY_MONTH	Публикация один раз в месяц
CALENDAR_FREQUENCY_QUARTER	Публикация один раз в квартал
CALENDAR_FREQUENCY_YEAR	Публикация один раз в год
CALENDAR_FREQUENCY_DAY	Публикация один раз в день

Тип события указывается в структуре `MqlCalendarEvent`. Возможные значения указаны в перечислении `ENUM_CALENDAR_EVENT_TYPE`

Идентификатор	Описание
CALENDAR_TYPE_EVENT	Событие (митинг, речь и так далее)
CALENDAR_TYPE_INDICATOR	Индикатор
CALENDAR_TYPE_HOLIDAY	Праздник

Сектор экономики, к которому относится событие, указывается в структуре `MqlCalendarEvent`. Возможные значения указаны в перечислении `ENUM_CALENDAR_EVENT_SECTOR`

Идентификатор	Описание
CALENDAR_SECTOR_NONE	Сектор не задан
CALENDAR_SECTOR_MARKET	Рынок, биржа
CALENDAR_SECTOR_GDP	Валовый внутренний продукт (GDP)
CALENDAR_SECTOR_JOBS	Рынок труда
CALENDAR_SECTOR_PRICES	Цены
CALENDAR_SECTOR_MONEY	Деньги
CALENDAR_SECTOR_TRADE	Торговля
CALENDAR_SECTOR_GOVERNMENT	Правительство
CALENDAR_SECTOR_BUSINESS	Бизнес
CALENDAR_SECTOR_CONSUMER	Потребление
CALENDAR_SECTOR_HOUSING	Жилье
CALENDAR_SECTOR_TAXES	Налоги
CALENDAR_SECTOR_HOLIDAYS	Праздники

Важность события указывается в структуре [MqlCalendarEvent](#). Возможные значения указаны в перечислении **ENUM\_CALENDAR\_EVENT\_IMPORTANCE**

Идентификатор	Описание
CALENDAR_IMPORTANCE_NONE	Степень важности не задана
CALENDAR_IMPORTANCE_LOW	Низкая важность
CALENDAR_IMPORTANCE_MODERATE	Средняя важность
CALENDAR_IMPORTANCE_HIGH	Высокая важность

Тип единицы измерения, в которых даются значения события, указывается в структуре [MqlCalendarEvent](#). Возможные значения указаны в перечислении **ENUM\_CALENDAR\_EVENT\_UNIT**

Идентификатор	Описание
CALENDAR_UNIT_NONE	Единица измерения не задана
CALENDAR_UNIT_PERCENT	Проценты
CALENDAR_UNIT_CURRENCY	Национальная валюта
CALENDAR_UNIT_HOUR	Количество часов
CALENDAR_UNIT_JOB	Количество рабочих мест

CALENDAR_UNIT_RIG	Буровые установки
CALENDAR_UNIT_USD	Доллары США
CALENDAR_UNIT_PEOPLE	Число людей
CALENDAR_UNIT_MORTGAGE	Количество ипотечных кредитов
CALENDAR_UNIT_VOTE	Число голосов
CALENDAR_UNIT_BARREL	Количество баррелях
CALENDAR_UNIT_CUBICFEET	Объем в кубических футах
CALENDAR_UNIT_POSITION	Чистый объем спекулятивных позиций в контрактах
CALENDAR_UNIT_BUILDING	Количество строений

В некоторых случаях значения экономического показателя требуют указания множителя, который указывается в структуре [MqlCalendarEvent](#). Возможные значения множителей указаны в перечислении **ENUM\_CALENDAR\_EVENT\_MULTIPLIER**

Идентификатор	Описание
CALENDAR_MULTIPLIER_NONE	Множитель не задан
CALENDAR_MULTIPLIER_THOUSANDS	Тысячи
CALENDAR_MULTIPLIER_MILLIONS	Миллионы
CALENDAR_MULTIPLIER_BILLIONS	Миллиарды
CALENDAR_MULTIPLIER_TRILLIONS	Триллионы

Потенциальное влияние события на курс национальной валюты указывается в структуре [MqlCalendarValue](#). Возможные значения указаны в перечислении **ENUM\_CALENDAR\_EVENT\_IMPACT**

Идентификатор	Описание
CALENDAR_IMPACT_NA	Влияния не указано
CALENDAR_IMPACT_POSITIVE	Положительное влияние
CALENDAR_IMPACT_NEGATIVE	Отрицательное влияние

Время наступления события указывается в структуре [MqlCalendarEvent](#). Возможные значения указаны в перечислении **ENUM\_CALENDAR\_EVENT\_TIMEMODE**

Идентификатор	Описание
---------------	----------

CALENDAR_TIMEMODE_DATETIME	Источник публикует точное время наступления события
CALENDAR_TIMEMODE_DATE	Событие занимает весь день
CALENDAR_TIMEMODE_NOTIME	Источник не публикует время события
CALENDAR_TIMEMODE_TENTATIVE	Источник не публикует предварительно точное время события, только его день. Время уточняется по факту наступления события

Смотри также

[Экономический календарь](#)

## Коды ошибок и предупреждений

Раздел содержит следующие описания:

- [Коды возврата торгового сервера](#) - анализ результатов отправки [торгового запроса](#), отправленного функцией [OrderSend\(\)](#);
- [Предупреждения компилятора](#) - коды предупредительных сообщений, выводимые при компиляции (не являются ошибками);
- [Ошибки компиляции](#) - коды сообщений об ошибке при неудачной попытке компиляции;
- [Ошибки времени выполнения](#) - коды ошибок при выполнении MQL5-программы, которые можно получить при помощи функции [GetLastError\(\)](#).

## Коды возврата торгового сервера

Все приказы на совершение торговых операций отправляются в виде структуры торгового запроса [MqlTradeRequest](#) с помощью функции [OrderSend\(\)](#). Результат выполнения этой функции помещается в структуру [MqlTradeResult](#), поле *retcode* которой содержит код возврата торгового сервера.

Код	Идентификатор	Описание
10004	TRADE_RETCODE_QUOTE	Реквота
10006	TRADE_RETCODE_REJECT	Запрос отклонен
10007	TRADE_RETCODE_CANCEL	Запрос отменен трейдером
10008	TRADE_RETCODE_PLACED	Ордер размещен
10009	TRADE_RETCODE_DONE	Заявка выполнена
10010	TRADE_RETCODE_DONE_PARTIAL	Заявка выполнена частично
10011	TRADE_RETCODE_ERROR	Ошибка обработки запроса
10012	TRADE_RETCODE_TIMEOUT	Запрос отменен по истечению времени
10013	TRADE_RETCODE_INVALID	Неправильный запрос
10014	TRADE_RETCODE_INVALID_VOLUME	Неправильный объем в запросе
10015	TRADE_RETCODE_INVALID_PRICE	Неправильная цена в запросе
10016	TRADE_RETCODE_INVALID_STOPS	Неправильные стопы в запросе
10017	TRADE_RETCODE_TRADE_DISABLED	Торговля запрещена
10018	TRADE_RETCODE_MARKET_CLOSED	Рынок закрыт
10019	TRADE_RETCODE_NO_MONEY	Нет достаточных денежных средств для выполнения запроса
10020	TRADE_RETCODE_PRICE_CHANGED	Цены изменились
10021	TRADE_RETCODE_PRICE_OFF	Отсутствуют котировки для обработки запроса
10022	TRADE_RETCODE_INVALID_EXPIRATION	Неверная дата истечения ордера в запросе

10023	TRADE_RETCODE_ORDER_CHANGED	Состояние ордера изменилось
10024	TRADE_RETCODE_TOO_MANY_REQUESTS	Слишком частые запросы
10025	TRADE_RETCODE_NO_CHANGES	В запросе нет изменений
10026	TRADE_RETCODE_SERVER_DISABLES_AT	Автотрейдинг запрещен сервером
10027	TRADE_RETCODE_CLIENT_DISABLES_AT	Автотрейдинг запрещен клиентским терминалом
10028	TRADE_RETCODE_LOCKED	Запрос заблокирован для обработки
10029	TRADE_RETCODE_FROZEN	Ордер или позиция заморожены
10030	TRADE_RETCODE_INVALID_FILE	Указан неподдерживаемый <a href="#">тип исполнения ордера</a> по остатку
10031	TRADE_RETCODE_CONNECTION	Нет соединения с торговым сервером
10032	TRADE_RETCODE_ONLY_REAL	Операция разрешена только для реальных счетов
10033	TRADE_RETCODE_LIMIT_ORDERS	Достигнут лимит на количество отложенных ордеров
10034	TRADE_RETCODE_LIMIT_VOLUME	Достигнут лимит на объем ордеров и позиций для данного символа
10035	TRADE_RETCODE_INVALID_ORDER	Неверный или запрещённый <a href="#">тип ордера</a>
10036	TRADE_RETCODE_POSITION_CLOSED	Позиция с указанным <a href="#">POSITION_IDENTIFIER</a> уже закрыта
10038	TRADE_RETCODE_INVALID_CLOSE_VOLUME	Закрываемый объем превышает текущий объем позиции
10039	TRADE_RETCODE_CLOSE_ORDER_EXIST	Для указанной позиции уже есть ордер на закрытие. Может возникнуть при работе в системе хеджинга: <ul style="list-style-type: none"> <li>• при попытке закрытия позиции встречной, если</li> </ul>

		<p>уже есть ордера на закрытие этой позиции</p> <ul style="list-style-type: none"> <li>• при попытке полного или частичного закрытия, если суммарный объем уже имеющихся ордеров на закрытие и вновь выставляемого ордера превышает текущий объем позиции</li> </ul>
10040	TRADE_RETCODE_LIMIT_POSITIONS	<p>Количество открытых позиций, которое можно одновременно иметь на счете, может быть ограничено настройками сервера. При достижении лимита в ответ на выставление ордера сервер вернет ошибку TRADE_RETCODE_LIMIT_POSITIONS. Ограничение работает по-разному в зависимости от типа учета позиций на счете:</p> <ul style="list-style-type: none"> <li>• Неттинговая система – учитывается количество открытых позиций. При достижении лимита платформа не позволит выставлять новые ордера, в результате исполнения которых может увеличиться количество открытых позиций. Фактически, платформа позволяет выставлять ордера только по тем символам, по которым уже есть открытые позиции. В неттинговой системе при проверке лимита не учитываются текущие отложенные ордера, поскольку их исполнение может привести к изменению текущих позиций, а не увеличению их количества.</li> <li>• Хеджинговая система – помимо открытых позиций, учитываются выставленные отложенные ордера,</li> </ul>

		поскольку их срабатывание всегда приводит к открытию новой позиции. При достижении лимита платформа не позволит выставлять рыночные ордера на открытие позиций, а также отложенные ордера.
10041	TRADE_RETCODE_REJECT_CANCEL	Запрос на активацию отложенного ордера отклонен, а сам ордер отменен
10042	TRADE_RETCODE_LONG_ONLY	Запрос отклонен, так как на символе установлено правило "Разрешены только длинные позиции" ( <a href="#">POSITION_TYPE_BUY</a> )
10043	TRADE_RETCODE_SHORT_ONLY	Запрос отклонен, так как на символе установлено правило "Разрешены только короткие позиции" ( <a href="#">POSITION_TYPE_SELL</a> )
10044	TRADE_RETCODE_CLOSE_ONLY	Запрос отклонен, так как на символе установлено правило "Разрешено только закрывать существующие позиции"
10045	TRADE_RETCODE_FIFO_CLOSE	Запрос отклонен, так как для торгового счета установлено правило "Разрешено закрывать существующие позиции только по правилу FIFO" ( <a href="#">ACCOUNT_FIFO_CLOSE=true</a> )

## Предупреждения компилятора

Предупреждения компилятора носят информационный характер и не являются сообщениями об ошибках.

Номер	Описание
21	Неполная запись даты в строке datetime
22	Ошибочные числа в строке datetime для даты, требования: год 1970<=X<=3000 месяц 0<X<=12 день 0<X<= 31/30/28(29)....
23	Ошибочные числа в строке datetime для времени, требования: час 0<=X<24 минута 0<=X<60
24	Некорректный цвет в формате RGB: одна из компонент RGB меньше 0 или больше 255
25	Неизвестный символ эскейп последовательности. Известные: \n \r \t \\ \" \' \'X \'x
26	Слишком большой объем локальных переменных (>512кб) функции, уменьшите их количество
29	Перечисление уже определено (дублирование) - члены будут добавлены к первому определению
30	Переопределение макроса
31	Переменная объявлена, но нигде не используется
32	Конструктор должен иметь тип void
33	Деструктор должен иметь тип void
34	Константа не вмещается в диапазон целых (X>_UI64_MAX    X<_I64_MIN) и будет преобразована в тип double
35	Слишком длинный HEX больше 16 значащих символов (обрезаются старшие полубайты)
36	Нет ни одного полубайта в HEX строке "0x"
37	Нет ни одной функции - нечего будет выполнять

38	Используется неинициализированная переменная
41	Функция не имеет тела, но и не вызывается
43	Возможны потери данных при преобразовании типа. Пример: int x=(double)z;
44	Потеря точности(данных) при преобразовании константы. Пример: int x=M_PI
45	Несовпадение знаков операндов в операциях сравнения. Пример: (char)c1>(uchar)c2
46	Проблемы с импортом функций - требуется объявление #import либо импорт функций уже открыт
47	Описание слишком большое - лишние символы не будут включены в исполняемый файл
48	Количество индикаторных буферов объявлено меньше, чем требуется
49	Не указан цвет для отрисовки графической серии в индикаторе
50	Нет ни одной графической серии для отображения индикатора
51	Не обнаружена функция-обработчик 'OnStart' в скрипте
52	Функция-обработчик 'OnStart' определена с неверными параметрами
53	Функция 'OnStart' может быть определена только в скрипте
54	Функция 'OnInit' определена с неверными параметрами
55	Функция 'OnInit' не используется в скриптах
56	Функция 'OnDeinit' определена с неверными параметрами
57	Функция 'OnDeinit' не используется в скриптах
58	Определены две функции 'OnCalculate'. Будет использована <a href="#">OnCalculate()</a> на одном ценовом массиве
59	Обнаружено переполнение при вычислении сложной <a href="#">целочисленной</a> константы
60	Возможно, переменная <a href="#">неинициализирована</a> .

61	Данное объявление делает недоступным обращение к <a href="#">локальной переменной</a> , объявленной на указанной строке
62	Данное объявление делает недоступным обращение к <a href="#">глобальной переменной</a> , объявленной на указанной строке
63	Не может быть использовано для <a href="#">статических массивов</a>
64	Данное объявление делает недоступным обращение к <a href="#">предопределенной переменной</a>
65	Значение выражения всегда <a href="#">true/false</a>
66	Использование переменной или выражения типа <a href="#">bool</a> в математических операциях является небезопасным
67	Результат применения оператора унарного минуса к беззнаковому типу <a href="#">ulong</a> неопределен
68	Версия, указанная в свойстве <a href="#">#property version</a> , недопустима для размещения в разделе <a href="#">Маркет</a> , правильный формат #property version "XXX.YYY"
69	Отсутствует выражение для выполнения по условию
70	Неверный возвращаемый тип функции или некорректные параметры при объявлении <a href="#">функции-обработчика события</a>
71	Требуется явное <a href="#">приведение структур</a> к одному типу
72	Данное объявление делает недоступным прямое обращение к члену <a href="#">класса</a> , объявленному на указанной строке. Доступ будет возможен только с помощью <a href="#">операции разрешения контекста ::</a>
73	Константа в двоичной записи слишком велика, старшие разряды будут отброшены
74	Параметр в методе <a href="#">наследуемого класса</a> отличается модификатором <a href="#">const</a> , дочерняя функция <a href="#">перегрузила</a> функцию родителя
75	Отрицательное или слишком большое значения смещения в <a href="#">битовой операции сдвига</a> , результат выполнения неопределен

76	Функция должна <a href="#">вернуть значение</a>
77	Функция типа <code>void</code> не должна возвращать значение
78	Не все варианты выполнения возвращают значение
79	Выражения на <a href="#">глобальном уровне</a> не разрешены
80	Возможна ошибка в <a href="#">последовательности выполнения операций</a> , используйте скобки для явного указания порядка
81	Найдено два вида вызова <a href="#">OnCalCulate()</a> . Вызываться будет вариант с использованием таймсерий OHLC
82	<a href="#">Структура</a> не содержит членов, размер будет приравнен 1 байту
83	Нет обработки результата выполнения функции
84	Индикатор, включаемый как ресурс, скомпилирован в режиме отладки. Это снижает его производительность. Для повышения скорости работы его нужно перекомпилировать
85	Слишком большой код символа в строке, должен быть в диапазоне от 0 до 65535
86	Нераспознанный служебный символ в строке
87	Не указано свойство индикатора, задающее вывод в главное окно или в отдельное подокно. Будет применено свойство <a href="#">#property indicator_chart_window</a>

## Ошибки компиляции

MetaEditor 5, редактор mq5-программ, выдает сообщения об ошибках программы, обнаруженных встроенным компилятором на стадии компиляции. Список этих ошибок приведен ниже в таблице. Для компиляции исходного кода в исполняемый нажмите **F7**. Программы с ошибками не могут быть скомпилированы, пока ошибки, указанные компилятором, не будут устранены.

Номер	Описание
100	Ошибка чтения файла
101	Ошибка открытия *.EX5 файла для записи
103	Недостаточно свободной памяти для завершения компиляции
104	Нераспознанная компилятором пустая синтаксическая единица
105	Некорректное имя файла в #include
106	Ошибка доступа к файлу в #include (возможно файл не существует)
108	Неподходящее имя для #define
109	Неизвестная команда препроцессора (допустимы #include,#define,#property,#import)
110	Неизвестный для компилятора символ
111	Функция не реализована (описание есть, тела нет)
112	Пропущена двойная кавычка ("")
113	Пропущена открывающая угловая скобка (<) или двойная кавычка ("")
114	Пропущена одинарная кавычка ('')
115	Пропущена закрывающая угловая скобка ">"
116	Не указан тип в объявлении
117	Нет оператора возврата return или имеется не во всех ветках выполнения
118	Ожидалась открывающая скобка параметров вызова
119	Ошибка записи EX5
120	Некорректный доступ к элементу массива
121	Функция не имеет тип void и оператор return должен вернуть значение

122	Некорректное объявление деструктора
123	Отсутствует двоеточие ":"
124	Переменная уже объявлена
125	Переменная с таким идентификатором уже объявлена
126	Имя переменной слишком длинное (>250 символов)
127	Структура с таким идентификатором уже определена
128	Структура не определена
129	Член структуры с таким именем уже определен
130	Нет такого члена структуры
131	Нарушена парность квадратных скобок
132	Ожидается открывающая круглая скобка "("
133	Несбалансированные фигурные скобки ( отсутствует "}" )
134	Сложно для компиляции (слишком большое ветвление, внутренний стек уровней переполнен)
135	Ошибка открытия файла на чтение
136	Недостаточно памяти для загрузки исходного файла в память
137	Ожидается переменная
138	Ссылка не может быть инициализирована
140	Ожидалось присваивание (возникает при объявлении)
141	Ожидается открывающая фигурная скобка "{"
142	Параметр может быть только <u>динамическим массивом</u>
143	Использование типа "void" недопустимо
144	Нет пары для ")" или "]", т.е. отсутствует "(" или "["
145	Нет пары для "(" или "[", т.е. отсутствует ")" или "]"
146	Некорректная размерность массива

147	Слишком много параметров (>64)
149	Этот токен тут не ожидается
150	Недопустимое использование операции (неправильные операнды)
151	Выражение типа void недопустимо
152	Ожидается оператор
153	Неправильное использование break
154	Ожидается точка с запятой ";"
155	Ожидается запятая ","
156	Тип должен быть определен как класс, а не как структура
157	Ожидалось выражение
158	В HEX встречается "не HEX символ" или слишком длинное число (количество цифр > 511)
159	Строка-константа имеет более 65534 символов
160	Определение функции здесь недопустимо
161	Неожиданный конец программы
162	Форвардная декларация для структур запрещена
163	Функция с таким именем уже определена и имеет иной тип возвращаемого значения
164	Функция с таким именем уже определена и имеет иной набор параметров
165	Функция с таким именем уже определена и реализована
166	Перегрузка функции для данного вызова не найдена
167	Функция с возвращаемым значением типа void не может возвращать значение
168	Функция не определена
170	Ожидается значение
171	В выражении case допустимы только целочисленные константы

172	Значение для case в этом switch уже использовано
173	Ожидается целочисленное значение
174	В выражении #import ожидается имя файла
175	Выражения на глобальном уровне не допустимы
176	Пропущена круглая скобка ")" перед ";"
177	Слева от знака равенства предполагается переменная
178	Результат выражения не используется
179	Объявление переменных в case недопустимо
180	Неявное преобразование из строки в число
181	Неявное преобразование числа в строку
182	Неоднозначный вызов перегруженной функции (подходят несколько перегрузок)
183	Недопустимый else без соответствующего if
184	Недопустимый case или default без соответствующего switch
185	Недопустимое использование эллипсиса
186	Инициализирующая последовательность имеет большее количество элементов чем инициализируемая переменная
187	Ожидается константа для case
188	Требуется константное выражение
189	Константная переменная не может быть изменена
190	Ожидается закрывающая скобка или запятая (объявление члена массива)
191	Идентификатор перечисления уже используется
192	Перечисление не может иметь модификаторов доступа (const, extern, static)
193	Член перечисления уже объявлен с другим значением
194	Существует переменная, определенная с таким же именем

195	Существует структура, определенная с таким же именем
196	Ожидается имя члена перечисления
197	Ожидается целочисленное выражение
198	Деление на ноль в константном выражении
199	Неверное количество параметров в функции
200	Параметром по ссылке должна быть переменная
201	Ожидается переменная такого же типа для передачи по ссылке
202	Константная переменная не может быть передана по неконстантной ссылке
203	Требуется целочисленная положительная константа
204	Ошибка доступа к защищенному члену класса
205	Импорт уже определен по другому пути
208	Исполняемый файл не создан
209	Для индикатора не найдена точка входа 'OnCalculate'
210	Оператор continue может быть использован только внутри цикла
211	Ошибка доступа к private(закрытому) члену класса
213	Метод структуры или класса не объявлен
214	Ошибка доступа к private(закрытому) методу класса
216	Копирование структур с объектами недопустимо
218	Выход индекса за границы массива
219	Недопустима инициализация массивов в объявлении структуры или класса
220	Конструктор класса не может иметь параметров
221	Деструктор класса не может иметь параметров
222	Метод класса или структуры с таким именем и параметрами уже объявлен

223	Ожидается операнд
224	Метод класса или структуры с таким именем есть, но с другими параметрами (объявление!=реализация)
225	Импортируемая функция не описана
226	Функция <a href="#">ZeroMemory()</a> не применима для классов с защищенными членами или наследованием
227	Неоднозначный вызов перегруженной функции (точное совпадение параметров для нескольких перегрузок)
228	Ожидается имя переменной
229	Ссылку нельзя объявить в этом месте
230	Уже используется в качестве имени перечисления
232	Ожидается класс или структура
235	Нельзя вызывать <code>delete</code> для удаления массива
236	Ожидается оператор 'while'
237	В <code>delete</code> должен быть указатель
238	<code>default</code> для этого <code>switch</code> уже есть
239	Синтаксическая ошибка
240	Escape-последовательность может встретиться только в строках ( начинается с '\' )
241	Требуется массив - квадратная скобка '[' не относится к массиву либо в качестве параметра-массива подают не массив
242	Не может быть инициализировано посредством инициализирующей последовательности
243	Импорт не определен
244	Ошибка оптимизатора на синтаксическом дереве
245	Объявлено слишком много структур (упростите программу)
246	Преобразование параметра недопустимо

247	Некорректное использование оператора delete
248	Нельзя объявить указатель на ссылку
249	Нельзя объявить ссылку на ссылку
250	Нельзя объявить указатель на указатель
251	Недопустимо объявление структуры в списке параметров
252	Недопустимая операция приведения типов
253	Указатель можно объявить только для класса или структуры
256	Необъявленный идентификатор
257	Ошибка оптимизатора исполняемого кода
258	Ошибка генерации исполняемого кода
260	Недопустимое выражение для оператора switch
261	Переполнение пула строковых констант, упростите программу
262	Невозможно преобразовать к перечислению
263	Нельзя использовать virtual для данных (членов класса или структуры)
264	Нельзя вызвать защищенный метод класса
265	Переопределяемая виртуальная функция возвращает другой тип
266	Класс нельзя наследовать от структуры
267	Структуру нельзя наследовать от класса
268	Конструктор не может быть виртуальным (спецификатор virtual недопустим)
269	Структура не может иметь виртуальных методов
270	Функция должна иметь тело
271	Перегрузка системных функций (функций терминала) запрещена
272	Спецификатор const недопустим для функций, не являющихся членом класса или структуры
274	Нельзя менять члены класса в константном методе

276	Неподходящая инициализирующая последовательность
277	Пропущено значение по умолчанию для параметра (специфика объявления параметров по умолчанию)
278	Переопределение параметра по умолчанию (в объявлении и реализации разные значения)
279	Нельзя вызвать неконстантный метод для константного объекта
280	Для доступа к членам требуется объект (поставлена точка для не класса/структурьи)
281	Имя уже объявленной структуры нельзя использовать при объявлении
284	Неразрешенное преобразование (при закрытом наследовании)
285	Структуры и массивы не могут быть использованы в качестве input-переменных
286	Спецификатор const недопустим для конструктора/деструктора
287	Неправильное строковое выражение для типа datetime
288	Неизвестное свойство (#property)
289	Некорректное значение для свойства
290	Некорректный индекс для свойства в #property
291	Пропущен параметр вызова - < func(x,) >
293	Объект должен быть передан по ссылке
294	Массив должен быть передан по ссылке
295	Функция была декларирована как экспортруемая
296	Функция не была декларирована как экспортруемая
297	Экспортовать импортируемую функцию нельзя
298	Импортируемая функция не может иметь такого параметра (нельзя передавать указатель, класс или структуру, содержащую динамический массив, указатель, класс и т.д.)

299	Должен быть класс
300	Секция #import не закрыта
302	Несоответствие типов
303	<u>extern</u> -переменная уже инициализирована
304	Не найдено ни одной <u>экспортируемой</u> функции или <u>стандартной точки входа</u>
305	Явный вызов <u>конструктора</u> запрещен
306	Метод был объявлен <u>константным</u>
307	Метод не был объявлен <u>константным</u>
308	Некорректный размер ресурсного файла
309	Некорректное имя ресурса
310	Ошибка открытия файла ресурса
311	Ошибка чтения файла ресурса
312	Неизвестный тип ресурса
313	Некорректный путь к файлу ресурса
314	Указанное имя <u>ресурса</u> уже используется
315	Ожидались параметры макроса
316	После имени макроса должен быть пробел
317	Ошибка в описании параметров макроса
318	Неверное число параметров при использовании макроса
319	Превышение максимального количества(16) параметров для макроса
320	Макрос слишком сложный, требуется упрощение
321	Параметром <u>EnumToString()</u> может быть только перечисление
322	Имя <u>ресурса</u> слишком длинное
323	Неподдерживаемый формат изображения (допустим только BMP-формат с глубиной цвета 24 или 32 бита)
324	Объявление массива внутри оператора запрещено
325	Функцию можно определить только на <u>глобальном</u> уровне

326	Данное объявление недопустимо для текущей <a href="#">области видимости</a> (области определения)
327	Инициализация статичных переменных значениями локальных недопустима
328	Недопустимое объявление массива объектов, не имеющих <a href="#">конструктора по умолчанию</a>
329	<a href="#">Список инициализации</a> разрешен только для <a href="#">конструкторов</a>
330	Отсутствует определение функции после <a href="#">списка инициализации</a>
331	<a href="#">Список инициализации</a> пуст
332	Инициализация массива в конструкторе запрещена
333	В <a href="#">списке инициализации</a> запрещено инициализировать члены родительского класса
334	Ожидалось выражение <a href="#">целого типа</a>
335	Требуемый объем памяти для <a href="#">массива</a> превышает максимально допустимое значение
336	Требуемый объем памяти для <a href="#">структур</a> превышает максимально допустимое значение
337	Требуемый объем памяти для переменных, объявленных на <a href="#">глобальном уровне</a> , превышает максимально допустимое значение
338	Требуемый объем памяти для <a href="#">локальных переменных</a> превышает максимально допустимое значение
339	<a href="#">Конструктор</a> не определен
340	Недопустимое имя для <a href="#">файла иконки</a>
341	Не удалось открыть <a href="#">файл иконки</a> по указанному пути
342	<a href="#">Файл иконки</a> некорректен и не соответствует формату <a href="#">ICO</a>
343	Повторная инициализация члена в конструкторе класса/структуре с помощью <a href="#">списка инициализации</a>

344	Инициализация <a href="#">статических</a> членов в <a href="#">списке инициализации</a> конструктора не допускается
345	Инициализация <a href="#">нестатического</a> члена класса/структуры на <a href="#">глобальном уровне</a> запрещена
346	Имя метода класса/структуры совпадает с ранее объявленным именем члена
347	Имя члена класса/структуры совпадает с ранее объявленным именем метода
348	<a href="#">Виртуальная</a> функция не может быть объявлена как <a href="#">static</a>
349	Модификатор <a href="#">const</a> недопустим для <a href="#">статической</a> функции
350	<a href="#">Конструктор</a> или <a href="#">деструктор</a> не могут быть статическими
351	Нельзя обращаться к нестатическому члену/методу класса или структуры из <a href="#">статической</a> функции
352	После ключевого слова <a href="#">operator</a> ожидается перегружаемая операция (+,-,[],++,-- и т.д.)
353	Не все операции можно <a href="#">перегружать</a> в MQL5
354	Определение не соответствует объявлению
355	Указано неверное количество параметров для <a href="#">оператора</a>
356	Не обнаружено ни одной <a href="#">функции-обработчика события</a>
357	Методы не могут быть <a href="#">экспортируемыми</a>
358	Нельзя приводить указатель на <a href="#">константный</a> объект к указателю на неконстантный объект
359	Шаблоны классов пока не поддерживаются
360	<a href="#">Перегрузка</a> шаблонов функций пока не поддерживается
361	Невозможно применить шаблон функции
362	Неоднозначный параметр в шаблоне функции (подходят несколько типов параметра)
363	Невозможно определить к какому типу параметра приводить аргумент шаблона функции

364	Неверное количество параметров в шаблоне функции
365	Шаблон функции не может быть <a href="#">виртуальным</a>
366	Шаблоны функций не могут быть экспортированы
367	Нельзя импортировать шаблоны функций
368	Структуры, содержащие объекты, недопустимы
369	Массивы строк и структуры, содержащие объекты, недопустимы
370	<a href="#">Статический член класса/структурь</a> должен быть явно инициализирован
371	Ограничение компилятора: строка не может содержать более 65 535 символов
372	Несогласованные <code>#ifdef/#endif</code>
373	Результатом выполнения функции не может быть объект класса, так как отсутствует конструктор копирования
374	Нельзя использовать нестатические члены и/или методы при инициализации статической переменной
375	OnTesterInit() нельзя использовать без объявления обработчика OnTesterDeinit()
376	Имя локальной переменной совпадает с именем одного из параметров функции
377	Нельзя использовать макросы <a href="#">_FUNCSIG</a> и <a href="#">_FUNCTION</a> вне тела функции
378	Недопустимый возвращаемый тип. Например, такая ошибка будет выдана для функций, импортированных из DLL, которые возвращают структуру или указатель в качестве результата
379	Ошибка при использовании шаблона
380	Не используется
381	Недопустимый синтаксис при объявлении чисто виртуальной функции, разрешено "=NULL" или "=0"
382	Только виртуальные функции могут быть объявлены со спецификатором чисто

		виртуальной функции ("=NULL" или "=0")
383		Нельзя создать экземпляр абстрактного класса
384		Для динамического приведения с помощью оператора <a href="#">dynamic_cast</a> типом назначения должен быть указатель на пользовательский тип
385		Ожидается тип "указатель на функцию"
386		Указатели на методы не поддерживаются
387		Ошибка - невозможно определить тип указателя на функцию
388		Приведение типа недоступно из-за <a href="#">закрытого</a> наследования
389		Переменная с модификатором <a href="#">const</a> должна быть проинициализирована при объявлении
393		В <a href="#">интерфейсе</a> могут быть объявлены только методы с <a href="#">публичным доступом</a>
394		Недопустимое вложение <a href="#">интерфейса</a> в другой интерфейс
395		Интерфейс может наследоваться только от другого интерфейса
396		Ожидается <a href="#">интерфейс</a>
397		Интерфейсы поддерживают только <a href="#">публичное наследование</a>
398		<a href="#">Интерфейс</a> не может содержать члены
399		Нельзя создавать объекты <a href="#">интерфейса</a> напрямую, только через наследование

## Ошибки времени выполнения

[GetLastError\(\)](#) - функция, возвращающая код последней ошибки, которая хранится в предопределенной переменной [LastError](#). Значение этой переменной можно сбросить в ноль функцией [ResetLastError\(\)](#).

Константа	Значение	Описание
ERR_SUCCESS	0	Операция выполнена успешно
ERR_INTERNAL_ERROR	4001	Неожиданная внутренняя ошибка
ERR_WRONG_INTERNAL_PARAMETER	4002	Ошибочный параметр при внутреннем вызове функции клиентского терминала
ERR_INVALID_PARAMETER	4003	Ошибочный параметр при вызове системной функции
ERR_NOT_ENOUGH_MEMORY	4004	Недостаточно памяти для выполнения системной функции
ERR_STRUCT_WITHOBJECTS_ORCLASS	4005	Структура содержит объекты строк и/или динамических массивов и/или структуры с такими объектами и/или классы
ERR_INVALID_ARRAY	4006	Массив неподходящего типа, неподходящего размера или испорченный объект динамического массива
ERR_ARRAY_RESIZE_ERROR	4007	Недостаточно памяти для перераспределения массива либо попытка изменения размера статического массива
ERR_STRING_RESIZE_ERROR	4008	Недостаточно памяти для перераспределения строки
ERR_NOTINITIALIZED_STRING	4009	Неинициализированная строка
ERR_INVALID_DATETIME	4010	Неправильное значение даты и/или времени
ERR_ARRAY_BAD_SIZE	4011	Общее число элементов в массиве не может превышать 2147483647
ERR_INVALID_POINTER	4012	Ошибочный указатель

ERR_INVALID_POINTER_TYPE	4013	Ошибочный тип указателя
ERR_FUNCTION_NOT_ALLOWE D	4014	Системная функция не разрешена для вызова
ERR_RESOURCE_NAME_DUPLIC ATED	4015	Совпадение имени <u>динамического</u> и <u>статического</u> ресурсов
ERR_RESOURCE_NOT_FOUND	4016	Ресурс с таким именем в EX5 не найден
ERR_RESOURCE_UNSUPPOTED _TYPE	4017	Неподдерживаемый тип ресурса или размер более 16 МВ
ERR_RESOURCE_NAME_IS_TO O_LONG	4018	Имя ресурса превышает 63 символа
ERR_MATH_OVERFLOW	4019	При вычислении математической функции произошло переполнение
<b>Графики</b>		
ERR_CHART_WRONG_ID	4101	Ошибочный идентификатор графика
ERR_CHART_NO_REPLY	4102	График не отвечает
ERR_CHART_NOT_FOUND	4103	График не найден
ERR_CHART_NO_EXPERT	4104	У графика нет эксперта, который мог бы обработать событие
ERR_CHART_CANNOT_OPEN	4105	Ошибка открытия графика
ERR_CHART_CANNOT_CHANG E	4106	Ошибка при изменении для графика символа и периода
ERR_CHART_WRONG_PARAME TER	4107	Ошибочное значение параметра для <u>функции по работе с графиком</u>
ERR_CHART_CANNOT_CREATE _TIMER	4108	Ошибка при создании таймера
ERR_CHART_WRONG_PROPER TY	4109	Ошибочный идентификатор свойства графика
ERR_CHART_SCREENSHOT_FAI LED	4110	Ошибка при создании скриншота
ERR_CHART_NAVIGATE_FAILE D	4111	Ошибка навигации по графику

ERR_CHART_TEMPLATE_FAILED	4112	Ошибка при применении шаблона
ERR_CHART_WINDOW_NOT_FOUND	4113	Подокно, содержащее указанный индикатор, не найдено
ERR_CHART_INDICATOR_CANNOT_ADD	4114	Ошибка при добавлении индикатора на график
ERR_CHART_INDICATOR_CANNOT_DEL	4115	Ошибка при удалении индикатора с графика
ERR_CHART_INDICATOR_NOT_FOUND	4116	Индикатор не найден на указанном графике
<b>Графические объекты</b>		
ERR_OBJECT_ERROR	4201	Ошибка при работе с графическим объектом
ERR_OBJECT_NOT_FOUND	4202	Графический объект не найден
ERR_OBJECT_WRONG_PROPERTY	4203	Ошибочный идентификатор свойства графического объекта
ERR_OBJECT_GETDATE_FAILED	4204	Невозможно получить дату, соответствующую значению
ERR_OBJECT_GETVALUE_FAILED	4205	Невозможно получить значение, соответствующее дате
<b>MarketInfo</b>		
ERR_MARKET_UNKNOWN_SYMBOL	4301	Неизвестный символ
ERR_MARKET_NOT_SELECTED	4302	Символ не выбран в MarketWatch
ERR_MARKET_WRONG_PROPERTY	4303	Ошибочный идентификатор свойства символа
ERR_MARKET_LASTTIME_UNKNOWN	4304	Время последнего тика неизвестно (тиков не было)
ERR_MARKET_SELECT_ERROR	4305	Ошибка добавления или удаления символа в MarketWatch
<b>Доступ к истории</b>		
ERR_HISTORY_NOT_FOUND	4401	Запрашиваемая история не найдена

ERR_HISTORY_WRONG_PROPERTY	4402	Ошибочный идентификатор свойства истории
ERR_HISTORY_TIMEOUT	4403	Превышен таймаут при запросе истории
ERR_HISTORY_BARS_LIMIT	4404	Количество запрашиваемых баров ограничено настройками терминала
ERR_HISTORY_LOAD_ERRORS	4405	Множество ошибок при загрузке истории
ERR_HISTORY_SMALL_BUFFER	4407	Принимающий массив слишком мал чтобы вместить все запрошенные данные
<b>Global_Variables</b>		
ERR_GLOBALVARIABLE_NOT_FOUND	4501	Глобальная переменная клиентского терминала не найдена
ERR_GLOBALVARIABLE_EXISTS	4502	Глобальная переменная клиентского терминала с таким именем уже существует
ERR_GLOBALVARIABLE_NOT_MODIFIED	4503	Не было модификаций глобальных переменных
ERR_GLOBALVARIABLE_CANT_READ	4504	Не удалось открыть и прочитать файл со значениями глобальных переменных
ERR_GLOBALVARIABLE_CANT_WRITE	4505	Не удалось записать файл со значениями глобальных переменных
ERR_MAIL_SEND_FAILED	4510	Не удалось отправить письмо
ERR_PLAY_SOUND_FAILED	4511	Не удалось воспроизвести звук
ERR_MQL5_WRONG_PROPERTY	4512	Ошибочный идентификатор свойства программы
ERR_TERMINAL_WRONG_PROPERTY	4513	Ошибочный идентификатор свойства терминала
ERR_FTP_SEND_FAILED	4514	Не удалось отправить файл по ftp
ERR_NOTIFICATION_SEND_FAILED	4515	Не удалось отправить <u>уведомление</u>

ERR_NOTIFICATION_WRONG_PARAMETER	4516	Неверный параметр для отправки уведомления - в функцию <a href="#">SendNotification()</a> передали пустую строку или <b>NULL</b>
ERR_NOTIFICATION_WRONG_SETTINGS	4517	Неверные настройки уведомлений в терминале (не указан ID или не выставлено разрешение)
ERR_NOTIFICATION_TOO_FREQUENT	4518	Слишком частая отправка уведомлений
ERR_FTP_NOSERVER	4519	Не указан FTP сервер
ERR_FTP_NOLOGIN	4520	Не указан FTP логин
ERR_FTP_FILE_ERROR	4521	Не найден файл в директории <code>MQL5\Files</code> для отправки на FTP сервер
ERR_FTP_CONNECT_FAILED	4522	Ошибка при подключении к FTP серверу
ERR_FTP_CHANGEDIR	4523	На FTP сервере не найдена директория для выгрузки файла
ERR_FTP_CLOSED	4524	Подключение к FTP серверу закрыто
Буферы пользовательских индикаторов		
ERR_BUFFERS_NO_MEMORY	4601	Недостаточно памяти для распределения индикаторных буферов
ERR_BUFFERS_WRONG_INDEX	4602	Ошибочный индекс своего индикаторного буфера
Свойства пользовательских индикаторов		
ERR_CUSTOM_WRONG PROPERTY	4603	Ошибочный идентификатор свойства пользовательского индикатора
Account		
ERR_ACCOUNT_WRONG_PROPERTY	4701	Ошибочный идентификатор свойства счета
ERR_TRADE_WRONG_PROPERTY	4751	Ошибочный идентификатор свойства торговли

ERR_TRADE_DISABLED	4752	Торговля для эксперта запрещена
ERR_TRADE_POSITION_NOT_FOUND	4753	Позиция не найдена
ERR_TRADE_ORDER_NOT_FOUND	4754	Ордер не найден
ERR_TRADE DEAL_NOT_FOUND	4755	Сделка не найдена
ERR_TRADE_SEND_FAILED	4756	Не удалось отправить торговый запрос
ERR_TRADE_CALC_FAILED	4758	Не удалось вычислить значение прибыли или маржи
Индикаторы		
ERR_INDICATOR_UNKNOWN_SYMBOL	4801	Неизвестный символ
ERR_INDICATOR_CANNOT_CREATE	4802	Индикатор не может быть создан
ERR_INDICATOR_NO_MEMORY	4803	Недостаточно памяти для добавления индикатора
ERR_INDICATOR_CANNOT_APPLY	4804	Индикатор не может быть применен к другому индикатору
ERR_INDICATOR_CANNOT_ADD	4805	Ошибка при добавлении индикатора
ERR_INDICATOR_DATA_NOT_FOUND	4806	Запрошенные данные не найдены
ERR_INDICATOR_WRONG_HANDLER	4807	Ошибочный хэндл индикатора
ERR_INDICATOR_WRONG_PARAMETERS	4808	Неправильное количество параметров при создании индикатора
ERR_INDICATOR_PARAMETERS_MISSING	4809	Отсутствуют параметры при создании индикатора
ERR_INDICATOR_CUSTOM_NAME	4810	Первым параметром в массиве должно быть имя пользовательского индикатора
ERR_INDICATOR_PARAMETER_TYPE	4811	Неправильный тип параметра в массиве при создании индикатора

ERR_INDICATOR_WRONG_INDEX	4812	Ошибочный индекс запрашиваемого индикаторного буфера
<b>Стакан цен</b>		
ERR_BOOKS_CANNOT_ADD	4901	Стакан цен не может быть добавлен
ERR_BOOKS_CANNOT_DELETE	4902	Стакан цен не может быть удален
ERR_BOOKS_CANNOT_GET	4903	Данные стакана цен не могут быть получены
ERR_BOOKS_CANNOT_SUBSCRIBE	4904	Ошибка при подписке на получение новых данных стакана цен
<b>Файловые операции</b>		
ERR_TOO_MANY_FILES	5001	Не может быть открыто одновременно более 64 файлов
ERR_WRONG_FILENAME	5002	Недопустимое имя файла
ERR_TOO_LONG_FILENAME	5003	Слишком длинное имя файла
ERR_CANNOT_OPEN_FILE	5004	Ошибка открытия файла
ERR_FILE_CACHEBUFFER_ERROR	5005	Недостаточно памяти для кеша чтения
ERR_CANNOT_DELETE_FILE	5006	Ошибка удаления файла
ERR_INVALID_FILEHANDLE	5007	Файл с таким хэндлом уже был закрыт, либо не открывался вообще
ERR_WRONG_FILEHANDLE	5008	Ошибочный хэндл файла
ERR_FILE_NOTTOWRITE	5009	Файл должен быть открыт для записи
ERR_FILE_NOTTOREAD	5010	Файл должен быть открыт для чтения
ERR_FILE_NOTBIN	5011	Файл должен быть открыт как бинарный
ERR_FILE_NOTTXT	5012	Файл должен быть открыт как текстовый
ERR_FILE_NOTTXTORCSV	5013	Файл должен быть открыт как текстовый или CSV

ERR_FILE_NOTCSV	5014	Файл должен быть открыт как CSV
ERR_FILE_READERROR	5015	Ошибка чтения файла
ERR_FILE_BINSTRINGSIZE	5016	Должен быть указан размер строки, так как файл открыт как бинарный
ERR_INCOMPATIBLE_FILE	5017	Для строковых массивов должен быть текстовый файл, для остальных - бинарный
ERR_FILE_IS_DIRECTORY	5018	Это не файл, а директория
ERR_FILE_NOT_EXIST	5019	Файл не существует
ERR_FILE_CANNOT_REWRITE	5020	Файл не может быть переписан
ERR_WRONG_DIRECTORYNAME	5021	Ошибочное имя директории
ERR_DIRECTORY_NOT_EXIST	5022	Директория не существует
ERR_FILE_ISNOT_DIRECTORY	5023	Это файл, а не директория
ERR_CANNOT_DELETE_DIRECTORY	5024	Директория не может быть удалена
ERR_CANNOT_CLEAN_DIRECTORY	5025	Не удалось очистить директорию (возможно, один или несколько файлов заблокированы и операция удаления не удалась)
ERR_FILE_WRITEERROR	5026	Не удалось записать ресурс в файл
ERR_FILE_ENDOFFILE	5027	Не удалось прочитать следующую порцию данных из CSV-файла (FileReadString, FileReadNumber, FileReadDatetime, FileReadBool), так как достигнут конец файла
Преобразование строк		
ERR_NO_STRING_DATE	5030	В строке нет даты
ERR_WRONG_STRING_DATE	5031	В строке ошибочная дата
ERR_WRONG_STRING_TIME	5032	В строке ошибочное время

ERR_STRING_TIME_ERROR	5033	Ошибка преобразования строки в дату
ERR_STRING_OUT_OF_MEMORY	5034	Недостаточно памяти для строки
ERR_STRING_SMALL_LEN	5035	Длина строки меньше, чем ожидалось
ERR_STRING_TOO_BIGNUMBER	5036	Слишком большое число, больше, чем ULONG_MAX
ERR_WRONG_FORMATSTRING	5037	Ошибочная форматная строка
ERR_TOO_MANY_FORMATTERS	5038	Форматных спецификаторов больше, чем параметров
ERR_TOO_MANY_PARAMETERS	5039	Параметров больше, чем форматных спецификаторов
ERR_WRONG_STRING_PARAMETER	5040	Испорченный параметр типа string
ERR_STRINGPOS_OUTOFRANGE	5041	Позиция за пределами строки
ERR_STRING_ZEROADDED	5042	К концу строки добавлен 0, бесполезная операция
ERR_STRING_UNKNOWNTYPE	5043	Неизвестный тип данных при конвертации в строку
ERR_WRONG_STRING_OBJECT	5044	Испорченный объект строки
<b>Работа с массивами</b>		
ERR_INCOMPATIBLE_ARRAYS	5050	Копирование несовместимых массивов. Строковый массив может быть скопирован только в строковый, а числовой массив - в числовой
ERR_SMALL_ASSERIES_ARRAY	5051	Приемный массив объявлен как AS_SERIES, и он недостаточного размера
ERR_SMALL_ARRAY	5052	Слишком маленький массив, стартовая позиция за пределами массива
ERR_ZEROSIZE_ARRAY	5053	Массив нулевой длины
ERR_NUMBER_ARRAYS_ONLY	5054	Должен быть числовой массив

ERR_ONEDIM_ARRAYS_ONLY	5055	Должен быть одномерный массив
ERR_SERIES_ARRAY	5056	Таймсериа не может быть использована
ERR_DOUBLE_ARRAY_ONLY	5057	Должен быть массив типа double
ERR_FLOAT_ARRAY_ONLY	5058	Должен быть массив типа float
ERR_LONG_ARRAY_ONLY	5059	Должен быть массив типа long
ERR_INT_ARRAY_ONLY	5060	Должен быть массив типа int
ERR_SHORT_ARRAY_ONLY	5061	Должен быть массив типа short
ERR_CHAR_ARRAY_ONLY	5062	Должен быть массив типа char
ERR_STRING_ARRAY_ONLY	5063	Должен быть массив типа string
<b>Работа с OpenCL</b>		
ERR_OPENCL_NOT_SUPPORTED	5100	<a href="#">Функции OpenCL</a> на данном компьютере не поддерживаются
ERR_OPENCL_INTERNAL	5101	Внутренняя ошибка при <a href="#">выполнении OpenCL</a>
ERR_OPENCL_INVALID_HANDLE	5102	Неправильный <a href="#">хэндл OpenCL</a>
ERR_OPENCL_CONTEXT_CREATE	5103	Ошибка при <a href="#">создании контекста OpenCL</a>
ERR_OPENCL_QUEUE_CREATE	5104	Ошибка создания очереди выполнения в OpenCL
ERR_OPENCL_PROGRAM_CREATE	5105	Ошибка при <a href="#">компиляции программы OpenCL</a>
ERR_OPENCL_TOO_LONG_KERNEL_NAME	5106	Слишком длинное имя точки входа ( <a href="#">кернел OpenCL</a> )
ERR_OPENCL_KERNEL_CREATE	5107	Ошибка создания кернел - точки входа <a href="#">OpenCL</a>
ERR_OPENCL_SET_KERNEL_PARAMETER	5108	Ошибка при <a href="#">установке параметров</a> для кернел OpenCL (точки входа в программу OpenCL)

ERR_OPENCL_EXECUTE	5109	Ошибка выполнения программы OpenCL
ERR_OPENCL_WRONG_BUFFER_SIZE	5110	Неверный размер буфера OpenCL
ERR_OPENCL_WRONG_BUFFER_OFFSET	5111	Неверное смещение в буфере OpenCL
ERR_OPENCL_BUFFER_CREATE	5112	Ошибка создания буфера OpenCL
ERR_OPENCL_TOO_MANY_OBJECTS	5113	Превышено максимальное число OpenCL объектов
ERR_OPENCL_SELECTDEVICE	5114	Ошибка выбора OpenCL устройства
<b>Работа с WebRequest()</b>		
ERR_WEBREQUEST_INVALID_ADDRESS	5200	URL не прошел проверку
ERR_WEBREQUEST_CONNECT_FAILED	5201	Не удалось подключиться к указанному URL
ERR_WEBREQUEST_TIMEOUT	5202	Превышен таймаут получения данных
ERR_WEBREQUEST_REQUEST_FAILED	5203	Ошибка в результате выполнения HTTP запроса
<b>Работа с сетью (сокетами)</b>		
ERR_NETSOCKET_INVALIDHANDLE	5270	В функцию передан неверный хэндл сокета
ERR_NETSOCKET_TOO_MANY_OPENED	5271	Открыто слишком много сокетов (максимум 128)
ERR_NETSOCKET_CANNOT_CONNECT	5272	Ошибка соединения с удаленным хостом
ERR_NETSOCKET_IO_ERROR	5273	Ошибка отправки/получения данных из сокета
ERR_NETSOCKET_HANDSHAKE_FAILED	5274	Ошибка установления защищенного соединения (TLS Handshake)
ERR_NETSOCKET_NO_CERTIFICATE	5275	Отсутствуют данные о сертификате, которым защищено подключение
<b>Пользовательские символы</b>		
ERR_NOT_CUSTOM_SYMBOL	5300	Должен быть указан пользовательский символ

ERR_CUSTOM_SYMBOL_WRONG_NAME	5301	Некорректное имя пользовательского символа. В имени символа можно использовать только латинские буквы без знаков препинания, пробелов и спецсимволов (допускаются ".", "_", "&" и "#"). Не рекомендуется использовать символы <, >, :, ", /, \,  , ?, *.
ERR_CUSTOM_SYMBOL_NAME_LONG	5302	Слишком длинное имя для пользовательского символа. Длина имени символа не должна превышать 32 знака с учётом завершающего 0
ERR_CUSTOM_SYMBOL_PATH_LONG	5303	Слишком длинный путь для пользовательского символа. Длина пути не более 128 знаков с учётом "Custom\\", имени символа, разделителей групп и завершающего 0
ERR_CUSTOM_SYMBOL_EXIST	5304	Пользовательский символ с таким именем уже существует
ERR_CUSTOM_SYMBOL_ERROR	5305	Ошибка при создании, удалении или изменении пользовательского символа
ERR_CUSTOM_SYMBOL_SELECTED	5306	Попытка удалить пользовательский символ, выбранный в обзоре рынка (Market Watch)
ERR_CUSTOM_SYMBOL_PROPERTY_WRONG	5307	Неправильное свойство пользовательского символа
ERR_CUSTOM_SYMBOL_PARAMETER_ERROR	5308	Ошибочный параметр при установке свойства пользователямского символа
ERR_CUSTOM_SYMBOL_PARAMETER_LONG	5309	Слишком длинный строковый параметр при установке свойства пользователямского символа
ERR_CUSTOM_TICKS_WRONG_ORDER	5310	Не упорядоченный по времени массив <a href="#">тиков</a>
Экономический календарь		

ERR_CALENDAR_MORE_DATA	5400	Размер массива недостаточен для получения описаний всех значений
ERR_CALENDAR_TIMEOUT	5401	Превышен лимит запроса по времени
ERR_CALENDAR_NO_DATA	5402	Страна не найдена
Пользовательские ошибки		
ERR_USER_ERROR_FIRST	65536	С этого кода начинаются ошибки, <a href="#">задаваемые пользователем</a>

Смотри также

[Коды возврата торгового сервера](#)

## Константы ввода/вывода

Константы:

- [Флаги открытия файлов](#)
- [Свойства файлов](#)
- [Позиционирование внутри файла](#)
- [Использование кодовой страницы](#)
- [MessageBox](#)

## Флаги открытия файлов

Значения флагов, определяющих режим работы с файлом. Флаги определены следующим образом:

Идентификатор	Значение	Описание
FILE_READ	1	Файл открывается для чтения. Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> ). При открытии файла обязательно должен быть указан флаг FILE_WRITE и/или флаг FILE_READ
FILE_WRITE	2	Файл открывается для записи. Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> ). При открытии файла обязательно должен быть указан флаг FILE_WRITE и/или флаг FILE_READ
FILE_BIN	4	Двоичный режим чтения-записи (без преобразования из строки в строку). Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> )
FILE_CSV	8	Файл типа csv (все записанные элементы преобразуются к строкам соответствующего типа, unicode или ansi, и разделяются разделителем). Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> )
FILE_TXT	16	Простой текстовый файл (тот же csv, однако разделитель не принимается во внимание). Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> )
FILE_ANSI	32	Строки типа ANSI (однобайтовые символы). Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> )
FILE_UNICODE	64	Строки типа UNICODE (двуихбайтовые символы). Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> )

FILE_SHARE_READ	128	Совместный доступ по чтению со стороны нескольких программ. Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> ), но не заменяет при открытии файла необходимости указать FILE_WRITE и/или флаг FILE_READ
FILE_SHARE_WRITE	256	Совместный доступ по записи со стороны нескольких программ. Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> ), но не заменяет при открытии файла необходимости указать FILE_WRITE и/или флаг FILE_READ
FILE_REWRITE	512	Возможность перезаписывания файла функциями <a href="#">FileCopy()</a> и <a href="#">FileMove()</a> . Файл должен существовать или открываться для записи. В противном случае файл открыт не будет
FILE_COMMON	4096	Расположение файла в общей папке всех клиентских терминалов \Terminal\Common\Files. Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> ), копировании файлов ( <a href="#">FileCopy()</a> , <a href="#">FileMove()</a> ) и проверке существования файлов ( <a href="#">FileIsExist()</a> )

При открытии файла можно указать один или более флагов, такое сочетание называется комбинацией флагов. Комбинация флагов записывается с помощью знака операции логического ИЛИ (|), который ставится между перечисляемыми флагами. Например, чтобы открыть файл в формате CSV одновременно на чтение и на запись, можно указать комбинацию FILE\_READ|FILE\_WRITE|FILE\_CSV.

#### Пример:

```
int filehandle=FileOpen(filename,FILE_READ|FILE_WRITE|FILE_CSV);
```

Есть некоторые особенности работы при указании флагов чтения и записи:

- Если указан FILE\_READ - делается попытка открытия уже существующего файла. Если файл не существует, то открыть файл не получится, новый файл не создается.

- Если FILE\_READ|FILE\_WRITE - создаётся новый файл если файл с таким именем отсутствует.
- Если FILE\_WRITE - файл создается заново с нулевым размером.

При открытии файла обязательно должен быть указан флаг FILE\_WRITE и/или флаг FILE\_READ.

Флаги, определяющие тип чтения открытого файла, имеют приоритет. Самый старший флаг FILE\_CSV, затем следует по старшинству флаг FILE\_BIN, и наименьший приоритет имеет флаг FILE\_TXT. Таким образом, если вдруг будут указаны сразу несколько флагов (FILE\_TXT|FILE\_CSV или FILE\_TXT|FILE\_BIN или FILE\_BIN|FILE\_CSV), то будет использован самый старший по приоритету флаг.

Флаги, определяющие тип кодировки, также имеют приоритет. Флаг FILE\_UNICODE имеет более старший приоритет, чем флаг FILE\_ANSI. Поэтому при указании комбинации FILE\_UNICODE|FILE\_ANSI будет использоваться флаг FILE\_UNICODE.

Если не указано ни FILE\_UNICODE, ни FILE\_ANSI, то подразумевается FILE\_UNICODE. Если не указано ни FILE\_CSV, ни FILE\_BIN, ни FILE\_TXT, то подразумевается FILE\_CSV.

Если файл открыт на чтение как текстовый (FILE\_TXT или FILE\_CSV), и при этом в начале файла будет обнаружен специальный двухбайтовый признак **0xff,0xfe**, то флаг кодировки будет FILE\_UNICODE, даже если указан флаг FILE\_ANSI.

Смотри также

[Файловые операции](#)

## Свойства файлов

Для получения свойств файлов используется функция [FileGetInteger\(\)](#), которой при вызове передается идентификатор требуемого свойства из перечисления ENUM\_FILE\_PROPERTY\_INTEGER

### ENUM\_FILE\_PROPERTY\_INTEGER

Идентификатор	Описание идентификатора
FILE_EXISTS	Проверка на существование
FILE_CREATE_DATE	Дата создания
FILE MODIFY_DATE	Дата последнего изменения
FILE_ACCESS_DATE	Дата последнего доступа к файлу
FILE_SIZE	Размер файла в байтах
FILE_POSITION	Позиция указателя в файле
FILE_END	Получение признака конца файла
FILE_LINE_END	Получение признака конца строки
FILE_IS_COMMON	Файл открыт в общей папке всех клиентских терминалов (смотри <a href="#">FILE_COMMON</a> )
FILE_IS_TEXT	Файл открыт как текстовый (смотри <a href="#">FILE_TXT</a> )
FILE_IS_BINARY	Файл открыт как бинарный (смотри <a href="#">FILE_BIN</a> )
FILE_IS_CSV	Файл открыт как CSV (смотри <a href="#">FILE_CSV</a> )
FILE_IS_ANSI	Файл открыт как ANSI (смотри <a href="#">FILE_ANSI</a> )
FILE_IS_READABLE	Файл открыт с возможностью чтения (смотри <a href="#">FILE_READ</a> )
FILE_IS_WRITABLE	Файл открыт с возможностью записи (смотри <a href="#">FILE_WRITE</a> )

Функция [FileGetInteger\(\)](#) имеет два варианта вызова. В первом варианте для получения свойств файла указывается его хэндл, полученный при открытии файла функцией [FileOpen\(\)](#). Этот вариант позволяет получить все свойства файла.

Второй вариант функции [FileGetInteger\(\)](#) возвращает значения свойств файла по его имени. В этом варианте можно получить только следующие общие свойства:

- FILE\_EXISTS - существование указанного по имени файла;
- FILE\_CREATE\_DATE - дата создания файла с указанным именем;
- FILE\_MODIFY\_DATE - дата изменения файла с указанным именем;
- FILE\_ACCESS\_DATE - дата последнего доступа к файлу с указанным именем;
- FILE\_SIZE - размер файла с указанным именем.

При попытке получения других свойств, кроме указанных выше, второй вариант вызова функции [FileGetInteger\(\)](#) вернет ошибку.

## Позиционирование внутри файла

Большая часть [файловых функций](#) связана с операциями чтения/записи информации. При этом с помощью функции [FileSeek\(\)](#) можно указывать положение файлового указателя на позицию внутри файла, с которой будет производится следующая операция чтения или записи. Перечисление **ENUM\_FILE\_POSITION** содержит допустимые положения указателя, относительно которого можно указать смещения в байтах для следующей операции.

### ENUM\_FILE\_POSITION

Идентификатор	Описание
SEEK_SET	Начало файла
SEEK_CUR	Текущая позиция файлового указателя
SEEK_END	Конец файла

Смотри также

[FileIsEnding](#), [FileIsLineEnding](#)

## Использование кодовой страницы в операциях преобразования строк

При операциях конвертации [строковых](#) переменных в массивы [типа char](#) и обратно в языке MQL5 используется кодировка, соответствующая по умолчанию текущей ANSI кодировке операционной системы Windows (CP\_ACP). Если требуется указать иной тип кодировки, то его можно задать дополнительным параметром для функций [CharArrayToString\(\)](#), [StringToCharArray\(\)](#) и [FileOpen\(\)](#).

В таблице приведены встроенные константы для некоторых наиболее востребованных кодовых страниц. Неперечисленные кодовые страницы можно указать кодом, соответствующим этой странице.

### Встроенные константы кодовых страниц

Константа	Значение	Описание
CP_ACP	0	Текущая кодовая страница ANSI кодировка в операционной системе Windows
CP_OEMCP	1	Текущая кодовая страница OEM.
CP_MACCP	2	Текущая кодовая страница Macintosh. <b>Примечание:</b> Это значение преимущественно используется в ранее созданных программных кодах и теперь в нем нет необходимости, так как современные компьютеры Macintosh используют Unicode кодировку.
CP_THREAD_ACP	3	Кодировка Windows ANSI для текущего потока выполнения.
CP_SYMBOL	42	Кодовая страница Symbol
CP_UTF7	65000	Кодовая страница UTF-7.
CP_UTF8	65001	Кодовая страница UTF-8.

### Смотри также

[Состояние клиентского терминала](#)

## Константы диалогового окна 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 OK
IDCANCEL	2	Выбрана кнопка Отмена (Cancel)
IDABORT	3	Выбрана кнопка Прервать (Abort)
IDRETRY	4	Выбрана кнопка Повтор (Retry)
IDIGNORE	5	Выбрана кнопка Пропустить (Ignore)
IDYES	6	Выбрана кнопка Да (Yes)
IDNO	7	Выбрана кнопка Нет (No)
IDTRYAGAIN	10	Выбрана кнопка Повторить (Try Again)
IDCONTINUE	11	Выбрана кнопка Продолжить (Continue)

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: OK. По умолчанию
MB_OKCANCEL	0x00000001	Окно сообщения содержит две кнопки: OK 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

## Программы MQL5

Для того чтобы mql5-программа могла работать, она должна быть скомпилирована (кнопка "Компилировать" или клавиша F7). Компиляция должна пройти без ошибок (допускаются предупреждения, которые необходимо проанализировать). При этом в соответствующей директории `terminal_dir\MQL5\Experts`, `terminal_dir\MQL5\indicators` или `terminal_dir\MQL5\scripts` должен быть создан выполняемый файл с тем же именем и расширением EX5. Именно этот файл может быть запущен на выполнение.

Особенности работы mql5-программ описаны в разделах:

- [Выполнение программ](#) - порядок вызова предопределенных функций-обработчиков событий;
- [Тестирование торговых стратегий](#) - особенности работы программ в тестере стратегий;
- [События клиентского терминала](#) - описание событий, которые можно обрабатывать в программах;
- [Вызов импортируемых функций](#) - порядок описания, допустимые параметры, порядок поиска и соглашения о связях импортируемых функций;
- [Ошибки выполнения](#) - получение информации об ошибках выполнения и критические ошибки.

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). MQL5-программы могут работать только при включенном клиентском терминале.

Для того чтобы эксперт прекратил свою работу, его необходимо удалить с графика. Для этого из контекстного меню графика следует выбрать "Список экспертов", далее выбрать советник из списка и нажать кнопку "Удалить". На работу советника также влияет состояние кнопки "Автоторговля".

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.

Сервисы для своей работы не требуют привязки к графику и предназначены для выполнения вспомогательных функций. Например, в сервисе можно создать [пользовательский символ](#), открыть график созданного символа, и затем в бесконечном цикле получать для него данные при помощи [сетевых функций](#) и непрерывно обновлять.

## Выполнение программ

Каждый скрипт, сервис и эксперт работает в собственном отдельном потоке. Все индикаторы, рассчитываемые на одном символе, даже если они запущены на разных графиках, работают в одном потоке. Таким образом, все индикаторы на одном символе делят между собой ресурсы одного потока.

В одном потоке с индикаторами также последовательно выполняются остальные действия по данному символу - обработка тиков и синхронизация истории. Это означает, что если в индикаторе выполняется бесконечное действие, все остальные события по его символу никогда не выполняются.

При запуске эксперта ему необходимо обеспечить актуальное [торговое окружение](#), [доступность истории](#) по данному символу и периоду, а также произвести [синхронизацию](#) между терминалом и сервером. На эти процедуры терминал предоставляет эксперту отсрочку запуска не более чем в 5 секунд, по истечении которых эксперт будет запущен с теми данными, что удалось подготовить. Поэтому при отсутствии связи с сервером это может привести к задержке запуска эксперта.

Краткая сводка по программам на MQL5 приведена в таблице:

Программа	Выполнение	Примечание
Сервис	В собственном потоке, сколько сервисов - столько потоков выполнения для них	Зацикленный сервис не может нарушить работу других программ
Скрипт	В собственном потоке, сколько скриптов - столько потоков выполнения для них	Зацикленный скрипт не может нарушить работу других программ
Эксперт	В собственном потоке, сколько экспертов - столько потоков выполнения для них	Зацикленный эксперт не может нарушить работу других программ
Индикатор	Один поток выполнения для всех индикаторов на одном символе. Сколько символов с индикаторами - столько потоков выполнения для них	Бесконечный цикл в одном индикаторе остановит работу всех остальных индикаторов на этом символе

Сразу после присоединения программы к графику производится ее загрузка в память клиентского терминала и [инициализация](#) глобальных переменных. Если какая-либо глобальная переменная типа класса имеет [конструктор](#), то этот конструктор будет вызван в процессе инициализации [глобальных переменных](#).

После этого программа находится в состоянии ожидания [события](#) от клиентского терминала. У каждой mq5-программы должна быть хотя бы одна [функция-обработчик](#) события, в противном случае загруженная программа выполниться не будет. Функции-обработчики событий имеют предопределенные имена, предопределенные наборы параметров и предопределенные типы возврата.

Тип	Имя функции	Параметры	Применение	Примечание
-----	-------------	-----------	------------	------------

int	<a href="#">OnInit</a>	нет	эксперты и индикаторы	Обработчик события <a href="#">Init</a> . Допускается тип возвращаемого значения void.
void	<a href="#">OnDeinit</a>	const int reason	эксперты и индикаторы	Обработчик события <a href="#">Deinit</a> .
void	<a href="#">OnStart</a>	нет	скрипты и сервисы	Обработчик события <a href="#">Start</a> .
int	<a href="#">OnCalculate</a>	const int rates_total, const int prev_calculated, const datetime &Time[], const double &Open[], const double &High[], const double &Low[], const double &Close[], const long &TickVolume[], const long &Volume[], const int &Spread[]	индикаторы	Обработчик события Calculate на всех ценовых данных.
int	<a href="#">OnCalculate</a>	const int rates_total, const int prev_calculated, const int begin, const double &price[]	индикаторы	Обработчик события <a href="#">Calculate</a> на одном массиве данных. В индикаторе не может одновременно присутствовать 2 обработчика Calculate. В этом случае будет работать только обработчик события Calculate на одном массиве данных.

void	<a href="#">OnTick</a>	нет	эксперты	Обработчик события <a href="#">NewTick</a> . Пока идет обработка события прихода нового тика другие события этого типа не приходят.
void	<a href="#">OnTimer</a>	нет	эксперты и индикаторы	Обработчик события <a href="#">Timer</a> .
void	<a href="#">OnTrade</a>	нет	эксперты	Обработчик события <a href="#">Trade</a> .
double	<a href="#">OnTester</a>	нет	эксперты	Обработчики события <a href="#">Tester</a>
void	<a href="#">OnChartEvent</a>	const int id, const long &lparam, const double &dparam, const string &sparam	эксперты и индикаторы	Обработчик события <a href="#">ChartEvent</a> .
void	<a href="#">OnBookEvent</a>	const string &symbol_name	эксперты и индикаторы	Обработчик события <a href="#">BookEvent</a> .

Клиентский терминал отсылает возникающие события в соответствующие открытые графики. Также события могут генерироваться графиками ([события графика](#)) либо mql5-программами ([пользовательские события](#)). Генерацию событий создания и удаления графических объектов на графике можно включать и отключать заданием свойств графика [CHART\\_EVENT\\_OBJECT\\_CREATE](#) и [CHART\\_EVENT\\_OBJECT\\_DELETE](#). Каждая mql5-программа и каждый график имеют свою собственную очередь событий, куда складываются все вновь поступающие события.

Программа получает события только от графика, на котором она запущена. Все события обрабатываются одно за другим в порядке поступления. Если в очереди уже есть событие [NewTick](#) либо это событие находится в состоянии обработки, то новое событие NewTick в очередь mql5-программы не ставится. Аналогично, если в очереди mql5-программы уже находится событие [ChartEvent](#) или такое событие обрабатывается, то новое событие такого типа не ставится в очередь. Обработка событий таймера производится по такой же схеме - если в очереди находится или уже обрабатывается событие [Timer](#), то новое событие таймера не ставится в очередь.

Очеди событий имеют ограниченный, но достаточный размер, поэтому переполнение очереди для корректно написанной программы маловероятно. При переполнении очереди новые события отбрасываются без постановки в очередь.

Крайне не рекомендуется использовать бесконечные циклы для обработки событий. Исключением из этого правила могут быть только скрипты и сервисы, которые обрабатывают одно единственное событие [Start](#).

[Библиотеки](#) не обрабатывают никаких событий.

## Запрет на использование функций в индикаторах и экспертах

Индикаторы, скрипты и эксперты являются исполняемыми программами на MQL5 и предназначены для различных типов задач. Поэтому существует ограничение на использование определенных функций в зависимости от [типа программы](#). В индикаторах запрещены следующие функции:

- [OrderCalcMargin\(\)](#);
- [OrderCalcProfit\(\)](#);
- [OrderCheck\(\)](#);
- [OrderSend\(\)](#);
- [SendFTP\(\)](#);
- [Sleep\(\)](#);
- [ExpertRemove\(\)](#);
- [MessageBox\(\)](#).

В свою очередь, в экспертах и скриптах запрещены все функции, предназначенные для индикаторов:

- [SetIndexBuffer\(\)](#);
- [IndicatorSetDouble\(\)](#);
- [IndicatorSetInteger\(\)](#);
- [IndicatorSetString\(\)](#);
- [PlotIndexSetDouble\(\)](#);
- [PlotIndexSetInteger\(\)](#);
- [PlotIndexSetString\(\)](#);
- [PlotIndexGetInteger](#).

Библиотека не является самостоятельной программой и выполняется в контексте вызвавшей её MQL5-программы: скрипт, индикатор или эксперт. Соответственно, на вызванную библиотеку распространяются указанные выше ограничения.

## Запрет на использование функций в сервисах

Сервисы не принимают никаких событий, так как не имеют привязки к графику. В сервисах запрещены следующие функции:

- [ExpertRemove\(\)](#);

[EventSetMillisecondTimer\(\)](#);  
[EventSetTimer\(\)](#);  
[EventKillTimer\(\)](#);  
[SetIndexBuffer\(\)](#);  
[IndicatorSetDouble\(\)](#);  
[IndicatorSetInteger\(\)](#);  
[IndicatorSetString\(\)](#);  
[PlotIndexSetDouble\(\)](#);  
[PlotIndexSetInteger\(\)](#);  
[PlotIndexSetString\(\)](#);  
[PlotIndexGetInteger\(\)](#);

## Загрузка и выгрузка индикаторов

Индикаторы загружаются в следующих случаях:

- прикрепление индикатора к графику;
- запуск терминала (если индикатор был прикреплен к графику перед предыдущим закрытием терминала);
- загрузка шаблона (если в шаблоне указан прикрепленный к графику индикатор);
- смена профиля (если индикатор прикреплен к одному из графиков профиля);
- смена символа и/или периода графика, к которому прикреплен индикатор;
- смена счета, к которому подключен терминал;
- после удачной перекомпиляции индикатора, если данный индикатор был прикреплен к графику.
- изменение [входных параметров](#) индикатора.

Индикаторы выгружаются в следующих случаях:

- при откреплении индикатора от графика;
- закрытие терминала (если индикатор был прикреплен к графику);
- загрузка шаблона, если к графику прикреплен индикатор;
- закрытие графика, к которому был прикреплен индикатор;
- смена профиля, если индикатор прикреплен к одному из графиков сменяемого профиля;
- смена символа и/или периода графика, к которому прикреплен индикатор;
- смена счета, к которому подключен терминал;
- изменение входных параметров индикатора.

## Загрузка и выгрузка экспертов

Загрузка эксперта производится в следующих случаях:

- прикрепление эксперта к графику;

- запуск терминала (если эксперт был прикреплен к графику перед предыдущим закрытием терминала);
- загрузка шаблона (если в шаблоне указан прикрепленный к графику эксперт);
- после удачной перекомпиляции эксперта, если данный эксперт был прикреплен к графику.
- смена профиля (если эксперт прикреплен к одному из графиков профиля);
- подключение к счету, даже если номер счета не менялся (если эксперт был прикреплен к графику перед авторизацией терминала на сервере).

Выгрузка эксперта, прикрепленного к графику, производится в следующих случаях:

- при откреплении эксперта от графика;
- при прикреплении эксперта к графику - если на данном графике был уже другой эксперт, то этот эксперт выгружается;
- закрытие терминала (если эксперт был прикреплен к графику);
- загрузка шаблона, если к графику прикреплен эксперт;
- закрытие графика, к которому был прикреплен эксперт;
- смена профиля, если эксперт прикреплен к одному из графиков сменяемого профиля;
- смена счета, к которому подключен терминал (если эксперт был прикреплен к графику перед авторизацией терминала на сервере);
- вызов функции [ExpertRemove\(\)](#).

При смене символа или таймфрейма графика, к которому эксперт прикреплен, выгрузка и загрузка эксперта не производится. При этом последовательно вызываются обработчики [OnDeinit\(\)](#) на старом символе/таймфрейме и [OnInit\(\)](#) на новом символе/таймфрейме (если они есть), значения глобальных переменных и [статических переменных](#) не сбрасываются. Все события, поступившие для эксперта до завершения инициализации (функции [OnInit\(\)](#)), пропускаются.

## Загрузка и выгрузка скриптов

Скрипты загружаются сразу после прикрепления к графику и выгружаются сразу после окончания своей работы. При этом функции [OnInit\(\)](#) и [OnDeinit\(\)](#) для скриптов не вызываются.

При выгрузке программы (удалении программы с графика) происходит деинициализация [глобальных](#) переменных и уничтожение очереди сообщений. В этом случае деинициализация означает освобождение переменных типа [string](#), освобождение [объектов динамических массивов](#) и вызов [деструкторов](#) при их наличии.

## Загрузка и выгрузка сервисов

Сервисы загружаются сразу после запуска терминала, если в момент остановки терминала они были запущены. Сервисы выгружаются сразу после окончания своей работы.

Сервисы имеют единственный обработчик [OnStart\(\)](#), в котором вы можете организовать бесконечный цикл получения и обработки данных, например - создание и обновление пользовательских символов при помощи сетевых функций.

В отличие от советников, индикаторов и скриптов, сервисы не привязаны к конкретному графику - поэтому для запуска сервиса предусмотрен отдельный механизм. Создание нового экземпляра сервиса производится из Навигатора с помощью команды "Добавить сервис". Для запуска, остановки и удаления экземпляра сервиса используйте его меню. Для управления всеми экземплярами, используйте меню самого сервиса.

Для лучшего понимания работы эксперта рекомендуется скомпилировать код приведенного в примере эксперта и произвести действия по загрузке/выгрузке экспертов, смене шаблона, символа, таймфрейма и так далее:

#### Пример:

```
//+-----+
//| TestExpert.mq5 |
//| Copyright 2009, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

class CTestClass
{
public:
    CTestClass() { Print("CTestClass constructor"); }
    ~CTestClass() { Print("CTestClass destructor"); }
};

CTestClass global;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//---
    Print("Initialization");
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//---
    Print("Deinitialization with reason ",reason);
}
//+-----+
//| Expert tick function |
//+-----+
```

```
//+-----+
void OnTick()
{
//---

}

//+-----+
```

Скрипты загружаются сразу после прикрепления к графику и выгружаются сразу после окончания своей работы.

#### Смотри также

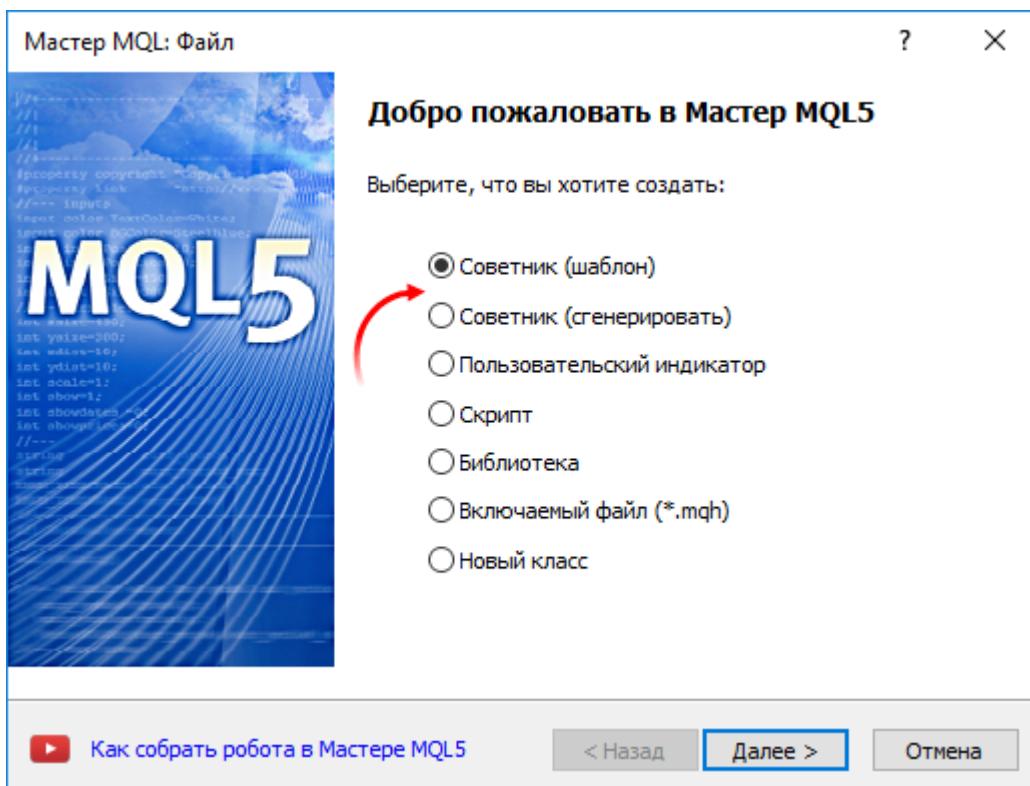
[События клиентского терминала](#), [Функции обработки событий](#)

## Разрешение на торговлю

### Автоматизация торговли

В языке MQL5 существует специальная группа [торговых функций](#), с помощью которых можно создавать автоматизированные торговые системы. Программы для автоматической торговли без участия человека называются экспертами (Expert Advisor) или торговыми роботами. Для создания эксперта в редакторе MetaEditor запустите мастер создания советников MQL5 Wizard и выберите один из двух вариантов:

- Expert Advisor (template) - позволяет создать шаблон с готовыми [функциями обработки событий](#), который необходимо дополнить всем необходимым функционалом с помощью самостоятельного программирования.
- Expert Advisor (generate) - дает возможность [создать полностью готового для работы торгового робота](#), просто выбирая необходимые вам модули: модуль формирования торговых сигналов, модуль управления капиталом и модуль подтягивания защитного стопа для открытых позиций (Trailing Stop).



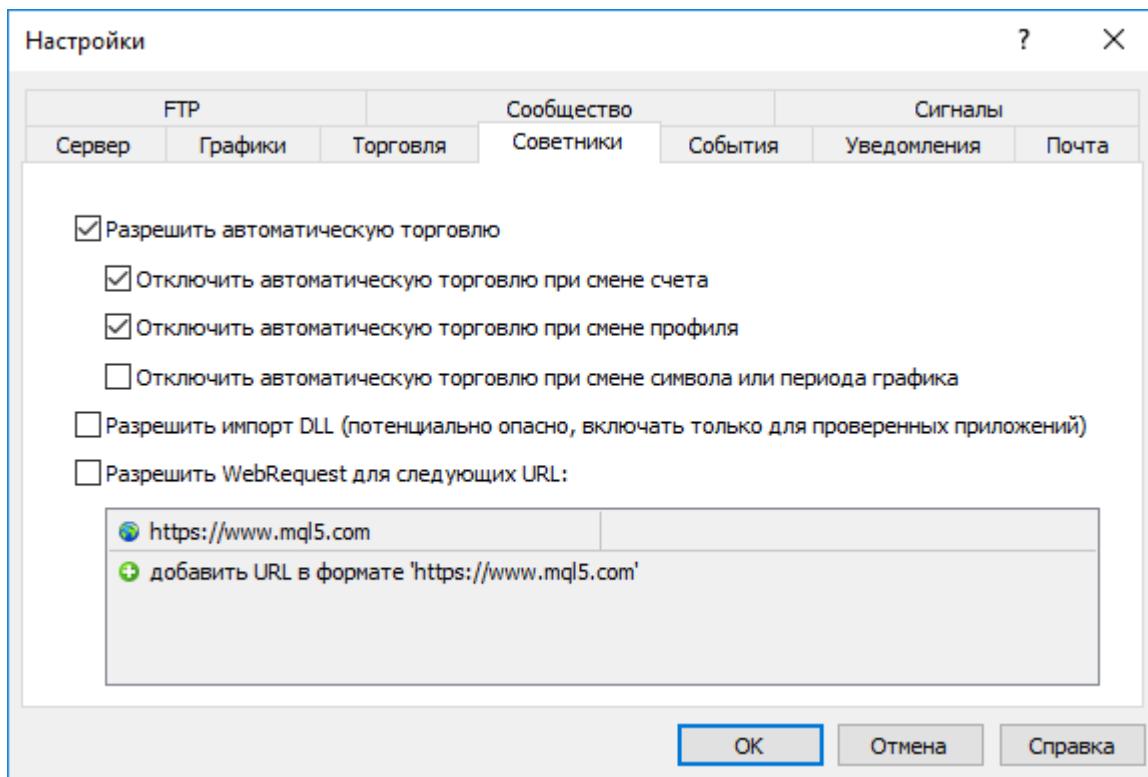
Торговые функции могут работать только в экспертах и скриптах, для индикаторов торговля запрещена.

### Проверка разрешения на автоматическую торговлю

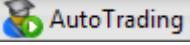
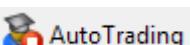
Для создания надежного эксперта, который мог бы работать в автоматическом режиме без контроля человека, необходимо организовать множество необходимых проверок. В первую очередь необходимо программным путем проверять, есть ли вообще разрешение на совершение торговых операций. Проверка разрешения на торговлю является базовой и обязательной при разработке любой автоматической системы.

## Проверка на разрешение автоматической торговли в самом терминале

В настройках терминала можно запретить или разрешить автоматическую торговлю для всех программ.



Переключать опцию разрешения автоматической торговли также можно прямо из Стандартной панели терминала:

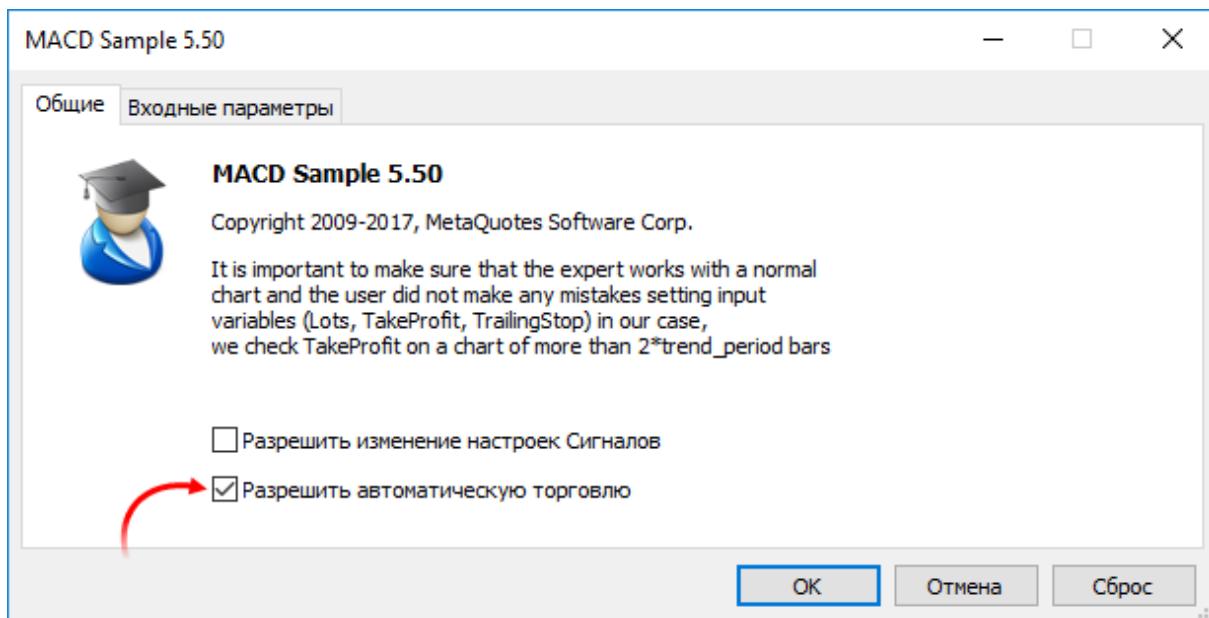
-  - автоматическая торговля разрешена, торговые функции в запущенных программах разрешены для использования.
-  - автоматическая торговля запрещена, при этом сами запущенные программы будут работать, хотя торговые функции не смогут выполняться.

Пример проверки:

```
if (!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
    Alert("Проверьте в настройках терминала разрешение на автоматическую торговлю!");
```

## Проверка разрешения на торговлю для данного запущенного эксперта/скрипта

При запуске программы можно разрешить или запретить автоматическую торговлю конкретно для нее. Для этого есть отдельная специальная настройка в свойствах программы.



Пример проверки:

```
if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
    Alert("Проверьте в настройках терминала разрешение на автоматическую торговлю!")
else
{
    if(!MQLInfoInteger(MQL_TRADE_ALLOWED))
        Alert("Автоматическая торговля запрещена в свойствах программы для ", __FILE__)
}
```

## Проверка разрешения на торговлю любым экспертам/скриптом для данного счета

Запрет на автоматическую торговлю может быть установлен на стороне торгового сервера. Пример проверки такой ситуации:

```
if(!AccountInfoInteger(ACCOUNT_TRADE_EXPERT))
    Alert("Автоматическая торговля запрещена для счета ", AccountInfoInteger(ACCOUNT_"
" на стороне торгового сервера");
```

Если для торгового счета запрещена автоматическая торговля, то торговые операции из экспертов/скриптов выполняться не будут.

## Проверка разрешения торговли для данного счета

Возможны случаи, когда для конкретного торгового счета запрещены любые торговые операции - нельзя торговать ни вручную, ни с помощью экспертов. Пример проверки ситуации, когда к торговому счету подключились с помощью инвесторского пароля:

```
if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED))
    Comment("Торговля запрещена для счета ", AccountInfoInteger(ACCOUNT_LOGIN),
    ".\n Возможно, подключение к торговому счету произведено по инвест паролю.
    "\n Проверьте журнал терминала, есть ли там такая запись:",
```

```
"\n\"",AccountInfoInteger(ACCOUNT_LOGIN),"\': trading has been disabled -
```

AccountInfoInteger(ACCOUNT\_TRADE\_ALLOWED) может возвращать `false` в следующих случаях:

- нет соединения с торговым сервером. Можно проверить с помощью TerminalInfoInteger(TERMINAL\_CONNECTED);
- торговый счет переведен в режим `read-only` (отправлен в архив);
- торговля на счете запрещена на стороне торгового сервера;
- подключение к торговому счету произведено в режиме инвестора.

#### Смотри также

[Состояние клиентского терминала](#), [Информация о счете](#), [Информация о запущенной MQL5-программе](#)

## События клиентского терминала

### **Init**

Сразу же после того, как клиентский терминал загрузит программу (эксперт или пользовательский индикатор) и запустит процесс инициализации глобальных переменных, будет послано событие **Init**, которое обрабатывается функцией [OnInit\(\)](#), если она есть. Это событие также генерируется 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 счета. Для скриптов событие **Init** не генерируется.

### **Deinit**

Перед деинициализацией глобальных переменных и выгрузкой программы (эксперт или пользовательский индикатор) клиентский терминал посыпает программе событие **Deinit**. Событие **Deinit** также генерируется 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 из параметра, переданного в функцию [OnDeinit\(\)](#). Выполнение функции [OnDeinit\(\)](#) ограничивается 2.5 секундами. Если за это время функция не закончила свою работу, то ее выполнение завершается принудительно. Для скриптов событие **Deinit** не генерируется.

### **Start**

Событие **Start** - это специальное событие для запуска скрипта или сервиса после его загрузки. Это событие обрабатывается функцией [OnStart](#). Событие **Start** экспертам и пользовательским индикаторам не посыпается.

### **NewTick**

Событие **NewTick** генерируется при поступлении новых котировок и обрабатывается функцией [OnTick\(\)](#) у присоединенных советников. Если при поступлении новой котировки выполнялась функция [OnTick](#), запущенная на предыдущей котировке, то пришедшая котировка будет проигнорирована советником, так как соответствующее событие не будет поставлено в очередь событий эксперта.

Все пришедшие во время выполнения программы новые котировки программой игнорируются до тех пор, пока не завершится очередное выполнение функции [OnTick\(\)](#). После этого функция будет запущена только после прихода очередной новой котировки.

Событие **NewTick** генерируется независимо от того, запрещена или разрешена автоматическая торговля (кнопка "Разрешить/запретить Авто-торговлю"). Запрет автоматической торговли означает только запрет на отправку торговых запросов из эксперта, работа эксперта не прекращается.

Запрет автоматической торговли путем нажатия на указанную кнопку не прерывает текущее выполнение функции [OnTick\(\)](#).

### **Calculate**

Событие [Calculate](#) генерируется только для индикаторов сразу после посылки события `Init` и при любом изменении ценовых данных. Обрабатывается функцией [OnCalculate](#).

### Timer

Событие [Timer](#) периодически генерируется клиентским терминалом для эксперта, который активизировал таймер при помощи функции [EventSetTimer](#). Обычно эта функция вызывается в функции `OnInit`. Обработка события `Timer` производится функцией [OnTimer](#). При завершении работы эксперта необходимо уничтожить созданный таймер при помощи [EventKillTimer](#), которую обычно вызывают в функции `OnDeinit`.

### Trade

Событие [Trade](#) генерируется при завершении торговой операции на торговом сервере. Обработка события `Trade` производится функцией [OnTrade\(\)](#) для следующих торговых операций:

- установка, модификация или удаление отложенного ордера;
- отмена отложенного ордера при нехватке средств либо по истечении срока действия;
- срабатывание отложенного ордера;
- открытие, добавление или закрытие позиции (или части позиции);
- модификация открытой позиции (изменение стопов).

### TradeTransaction

В результате выполнения определенных действий с торговым счетом, его состояние изменяется. К таким действиям относятся:

- Отсылка торгового запроса любым MQL5-приложением в клиентском терминале при помощи функций [OrderSend](#) и [OrderSendAsync](#) и его последующее исполнение;
- Отсылка торгового запроса через графический интерфейс терминала и его последующее исполнение;
- Срабатывание отложенных ордеров и стоп-ордеров на сервере;
- Выполнение операций на стороне торгового сервера.

В результате данных действий, для счета выполняются торговые транзакции:

- обработка торгового запроса;
- изменение открытых ордеров;
- изменение истории ордеров;
- изменение истории сделок;
- изменение позиций.

Например, при отсылке рыночного ордера на покупку, он обрабатывается, для счета создается соответствующий ордер на покупку, происходит исполнение ордера, его удаление из списка открытых, добавление в историю ордеров, далее добавляется соответствующая сделка в историю и создается новая позиция. Все эти действия являются торговыми транзакциями. Приход каждой такой транзакции в терминал является событием `TradeTransaction`. Данное событие обрабатывается функцией [OnTradeTransaction](#).

### Tester

Событие [Tester](#) генерируется по окончании тестирования эксперта на исторических данных. Обработка события Tester производится функцией [OnTester\(\)](#).

### **TesterInit**

Событие [TesterInit](#) генерируется при запуске оптимизации в тестере стратегий перед самым первым проходом. Обработка события TesterInit производится функцией [OnTesterInit\(\)](#).

### **TesterPass**

Событие [TesterPass](#) генерируется при поступлении нового [фрейма данных](#). Обработка события TesterPass производится функцией [OnTesterPass\(\)](#).

### **TesterDeinit**

Событие [TesterDeinit](#) генерируется по окончании оптимизации эксперта в тестере стратегий. Обработка события TesterDeinit производится функцией [OnTesterDeinit\(\)](#).

### **ChartEvent**

Событие [ChartEvent](#) генерируется клиентским терминалом при работе пользователя с графиком:

- нажатия клавиатуры, когда окно графика находится в фокусе;
- создание [графического объекта](#);
- удаление [графического объекта](#);
- щелчок мыши на графическом объекте, принадлежащего графику;
- перемещение графического объекта при помощи мыши;
- окончание редактирования текста в поле ввода графического объекта LabelEdit.

Также существует пользовательское событие ChartEvent, которое может послать эксперту любая mql5-программа при помощи функции [EventChartCustom](#). Событие обрабатывается функцией [OnChartEvent](#).

### **BookEvent**

Событие [BookEvent](#) генерируется клиентским терминалом при изменении состояния стакана цен и обрабатывается функцией [OnBookEvent](#). Для того чтобы клиентский терминал начал генерировать события BookEvent по конкретному символу, достаточно предварительно подписаться на получение этих событий для этого символа с помощью функции [MarketBookAdd](#).

Для того чтобы отписаться от получения события BookEvent по символу, необходимо вызывать функцию [MarketBookRelease](#). Событие BookEvent является широковещательным - это означает, что достаточно одному эксперту подписаться на получение события BookEvent с помощью функции MarketBookAdd, и все остальные эксперты, имеющие обработчик OnBookEvent, будут получать это событие. Поэтому необходимо анализировать имя символа, которое передается в обработчик в качестве параметра.

### **Смотри также**

[Функции обработки событий](#), [Выполнение программ](#)

## Ресурсы

### Использование графики и звука в программах на MQL5

Программы на MQL5 позволяют работать со звуковыми и графическими файлами:

- [PlaySound\(\)](#) воспроизводит звуковой файл;
- [ObjectCreate\(\)](#) позволяет с помощью [графических объектов](#) OBJ\_BITMAP и OBJ\_BITMAP\_LABEL создавать пользовательские интерфейсы.

#### PlaySound()

Пример вызова функции [PlaySound\(\)](#):

```
//+-----+
//| функция вызывает штатную OrderSend() и воспроизводит звук |
//+-----+
void OrderSendWithAudio(MqlTradeRequest &request, MqlTradeResult &result)
{
    //--- отправим запрос на сервер
    OrderSend(request, result);
    //--- если запрос принят, играем звук Ok.wav
    if(result.retcode==TRADE_RETCODE_PLACED)
        PlaySound("Ok.wav");
    //--- при неудаче выдаем тревожный звук из файла timeout.wav
    else
        PlaySound("timeout.wav");
}
```

В данном примере показано как проигрывать звуки из файлов Ok.wav и timeout.wav, входящих в стандартную поставку терминала. Эти файлы находятся в папке **каталог\_терминала\Sounds**. Здесь **каталог\_терминала** означает папку, из которой запущен клиентский терминал MetaTrader 5. Программным путем из mql5-программы каталог терминала можно узнать следующим образом:

```
//--- Папка, в которой хранятся данные терминала
string terminal_path=TerminalInfoString(TERMINAL_PATH);
```

Можно использовать звуковые файлы не только из папки **каталог\_терминала\Sounds**, но и из любой подпапки, находящейся в папке **каталог\_данных\_терминала\MQL5**. Расположение каталога данных терминала на компьютере можно выяснить через меню терминала "Файл"->"Открыть каталог данных" или программным путем:

```
//--- Папка, в которой хранятся данные терминала
string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
```

Например, если звуковой файл Demo.wav лежит в папке **каталог\_данных\_терминала\MQL5\Files**, то вызов PlaySound() должен быть записан таким образом:

```
//--- проиграем звуковой файл Demo.wav из папки каталог_данных_терминала\MQL5\Files\
PlaySound("\\Files\\Demo.wav");
```

Обратите внимание на то, что в комментарии путь к файлу написан с использованием символа "\", а в самой функции для разделения папок в пути используется последовательность "\\\".

При указании пути всегда используйте только двойную обратную косую черту в качестве разделителя, так как одиночная обратная черта является управляемым символом для компилятора при разборе константных строк и [символьных констант](#) в исходном коде программы.

Для остановки воспроизведения файла нужно вызывать функцию [PlaySound\(\)](#) с параметром NULL:

```
//--- вызов PlaySound() с параметром NULL останавливает воспроизведение звука
PlaySound(NULL);
```

## ObjectCreate()

Пример эксперта, который с помощью функции [ObjectCreate\(\)](#) создает объект "Графическая метка" (OBJ\_BITMAP\_LABEL).

```
string label_name="currency_label";           // имя объекта OBJ_BITMAP_LABEL
string euro        ="\\Images\\euro.bmp";      // путь к файлу каталог_данных_терминала\MQ5\Experts\currency.mq5\Images\euro.bmp
string dollar      ="\\Images\\dollar.bmp";    // путь к файлу каталог_данных_терминала\MQ5\Experts\currency.mq5\Images\dollar.bmp
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- создадим кнопку OBJ_BITMAP_LABEL, если ее еще нет
if(ObjectFind(0,label_name)<0)
{
//--- попробуем создать объект OBJ_BITMAP_LABEL
bool created=ObjectCreate(0,label_name,OBJ_BITMAP_LABEL,0,0,0);
if(created)
{
//--- привяжем кнопку к правому верхнему углу графика
ObjectSetInteger(0,label_name,OBJPROP_CORNER,CORNER_RIGHT_UPPER);
//--- теперь настроим свойства объекта
ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,100);
ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,50);
//--- сбросим код последней ошибки в 0
ResetLastError();
//--- загрузим картинку для состояния кнопки "Нажата"
bool set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,0,euro);
//--- проверим результат
if(!set)
{
PrintFormat("Не удалось загрузить картинку из файла %s. Код ошибки %d",euro,GetLastError());
}
ResetLastError();
//--- загрузим картинку для состояния кнопки "Отжата"
```

```

set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,1,dollar);

if(!set)
{
    PrintFormat("Не удалось загрузить картинку из файла %s. Код ошибки %d",do
}
//--- Отдадим графику команду на обновление, чтобы кнопка появилась сразу же,
ChartRedraw(0);
}

else
{
//--- объект создать не удалось, сообщим об этом
PrintFormat("Не удалось создать объект OBJ_BITMAP_LABEL. Код ошибки %d",GetLa
}

}

//---

return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- удалим объект с графика
ObjectDelete(0,label_name);
}

```

Создание и настройка графического объекта с именем currency\_label происходят в функции OnInit(). Пути к загружаемым графическим файлам заданы в [глобальных переменных euro](#) и [dollar](#), в качестве разделителя использована двойная обратная косая черта:

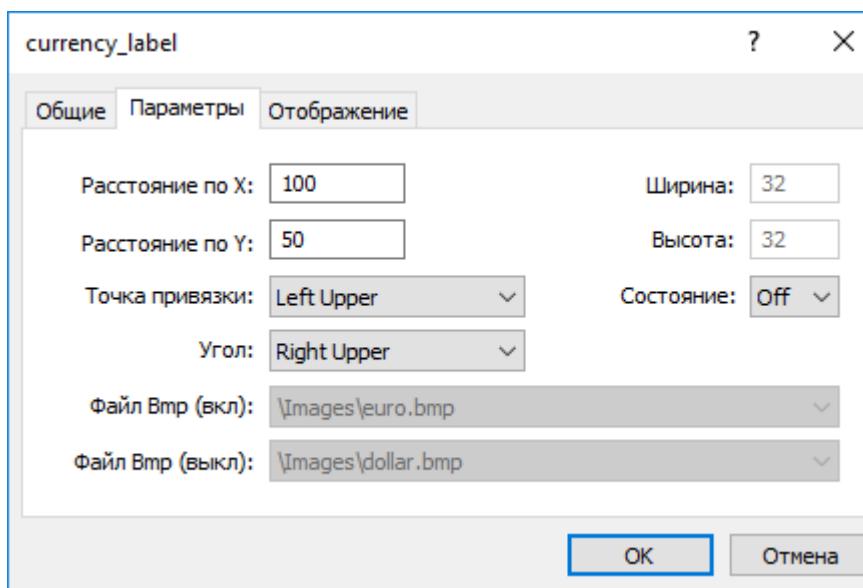
```

string euro      ="\\Images\\euro.bmp";      // путь к файлу каталог_данных_терминала\MQ
string dollar    ="\\Images\\dollar.bmp";    // путь к файлу каталог_данных_терминала\MQ

```

Сами файлы при этом находятся в папке **каталог\_данных\_терминала\МQL5\Images**.

Объект OBJ\_BITMAP\_LABEL фактически представляет собою кнопку, которая в зависимости от состояния (нажата или отжата) может отображать одну из двух картинок: euro.bmp или dollar.bmp.



Размеры кнопки с графическим интерфейсом автоматически устанавливаются под размер отображаемой картинки. Смена изображения производится при нажатии левой кнопкой мышки на объекте OBJ\_BITMAP\_LABEL (в свойствах должна быть выбрана опция "Отключить выделение"). Объект OBJ\_BITMAP создается аналогичным образом и предназначен для создания фона с требуемым рисунком.

Значение свойства [OBJPROP\\_BMPFILE](#), которое отвечает за вид объектов OBJ\_BITMAP и OBJ\_BITMAP\_LABEL, можно менять динамически. Это позволяет создавать разнообразные интерактивные пользовательские интерфейсы для mq5-программ.

## Включение ресурсов в исполняемые файлы при компиляции mq5-программ

Для работы mq5-программы может потребоваться множество разнообразных загружаемых ресурсов в виде файлов изображений и звуков. Для того чтобы исключить необходимость переноса всех этих файлов при передаче исполняемой программы на MQL5, следует использовать директиву компилятора `#resource`:

```
#resource путь_к_файлу_ресурса
```

Команда `#resource` указывает компилятору, что ресурс по указанному пути `путь_к_файлу_ресурса` нужно включить в исполняемый файл EX5. Таким образом, все необходимые картинки и звуки можно поместить непосредственно в EX5-файл и для запуска программы в другом терминале не потребуется передавать все используемые в ней отдельные файлы. Любой EX5-файл может содержать ресурсы, и любая EX5-программа может использовать ресурсы из другой EX5-программы.

Файлы в формате BMP и WAV перед включением в исполняемый EX5 файл автоматически сжимаются. Это означает, что использование ресурсов не только позволяет создавать полноценные программы на MQL5, но и уменьшает общий размер требуемых терминалу файлов при использовании графики и звука по сравнению с обычным способом написания mq5-программ.

Размер файла ресурса не может быть больше 128 Mb.

## Поиск указанных ресурсов компилятором

Ресурс вставляется командой #resource "<путь к файлу ресурса>"

```
#resource "<путь_к_файлу_ресурса>"
```

Длина константной строки <путь\_к\_файлу\_ресурса> не должна превышать 63 символа.

Компилятор ищет ресурс по указанному пути в следующей последовательности:

- если в начале пути стоит разделитель обратная косая черта "\" ( пишется "\\"), то ресурс ищется относительно каталога **каталог\_данных\_терминала\МQL5\**,
- если обратной косой черты нет, то ресурс ищется относительно расположения исходного файла, в котором этот ресурс прописывается.

В пути ресурса недопустимо использовать подстроки "..\\\" и ":\\\".

Примеры включения ресурсов:

```
//--- правильное указание ресурсов
#resource "\\Images\\euro.bmp" // euro.bmp находится в каталог_данных_терминала\МQL5\
#resource "picture.bmp"      // picture.bmp находится в том же каталоге, где и исходный файл
#resource "Resource\\map.bmp" // ресурс находится в папке каталог_исходного_файла\Res

//--- неправильное указание ресурсов
#resource ":picture_2.bmp"    // нельзя использовать ":" 
#resource "..\\picture_3.bmp"  // нельзя использовать ".."
#resource "\\Files\\Images\\Folder_First\\My_panel\\Labels\\too_long_path.bmp" //большой путь
```

## Использование ресурсов

### Имя ресурса

После того как ресурс объявлен директивой **#resource**, его можно использовать в любой части программы. Именем ресурса становится его путь без косой черты в начале строки, задающей путь к ресурсу. Для использования своего ресурса в коде нужно перед именем ресурса добавлять специальный признак "::".

Примеры:

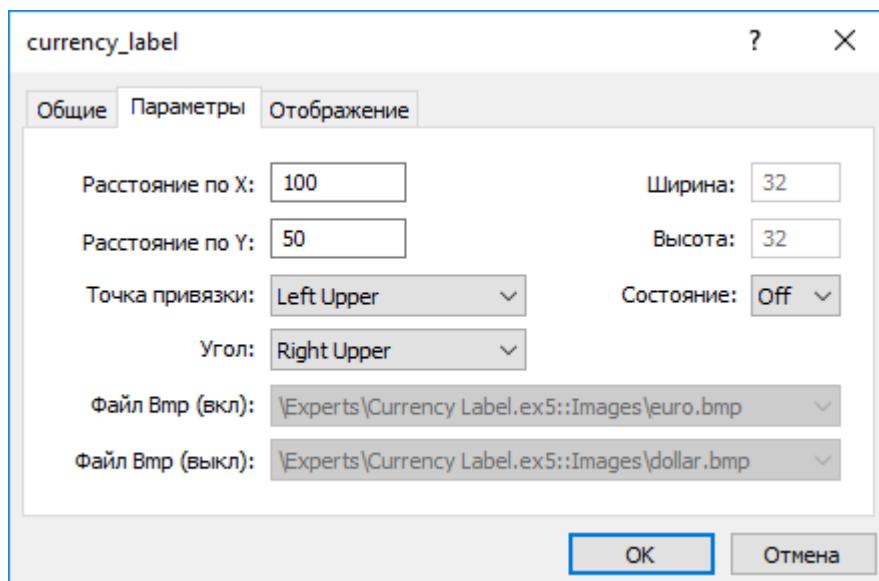
```
//--- примеры указания ресурсов и их имена в комментарии
#resource "\\Images\\euro.bmp"           // имя ресурса - Images\euro.bmp
#resource "picture.bmp"                 // имя ресурса - picture.bmp
#resource "Resource\\map.bmp"           // имя ресурса - Resource\map.bmp
#resource "\\Files\\Pictures\\good.bmp" // имя ресурса - Files\Pictures\good.bmp
#resource "\\Files\\Demo.wav";          // имя ресурса - Files\Demo.wav"
#resource "\\Sounds\\thrill.wav";       // имя ресурса - Sounds\thrill.wav"
...
```

```
//--- использование ресурсов
ObjectSetString(0,bitmap_name,OBJPROP_BMPFILE,0,"::Images\\euro.bmp");
...
ObjectSetString(0,my_bitmap,OBJPROP_BMPFILE,0,"::picture.bmp");
...
set=ObjectSetString(0,bitmap_label,OBJPROP_BMPFILE,1,"::Files\\Pictures\\good.bmp");
...
PlaySound("::Files\\Demo.wav");
...
PlaySound("::Sounds\\thrill.wav");
```

Необходимо отметить, что при установке объектам OBJ\_BITMAP и OBJ\_BITMAP\_LABEL изображения из ресурса, значение свойства OBJPROP\_BMPFILE уже нельзя менять вручную. Например, пусть мы используем для создания OBJ\_BITMAP\_LABEL ресурсы файлов euro.bmp и dollar.bmp.

```
#resource "\\Images\\euro.bmp"; // euro.bmp находится в каталог_данных_терминала\М
#resource "\\Images\\dollar.bmp"; // dollar.bmp находится в каталог_данных_терминала\
```

Тогда при просмотре свойств этого объекта мы увидим, что свойства BitMap File (On) и BitMap File (Off) имеют серый цвет и недоступны для изменения вручную:



## Использование ресурсов других mql5-программ

Использование ресурсов имеет и другое преимущество - в любой mql5-программе можно использовать ресурсы из любого файла EX5. Таким образом, ресурсы из одного файла EX5 можно использовать во многих других mql5-программах.

Для того чтобы использовать имя ресурса из стороннего файла, его нужно указать в виде <путь\_имя\_файла\_EX5>::<имя\_ресурса>. Например, пусть в скрипте Draw\_Triangles\_Script.mq5 указан ресурс на картинку в файле triangle.bmp:

```
#resource "\\Files\\triangle.bmp"
```

Тогда его имя для использования в самом скрипте будет выглядеть как "Files\triangle.bmp", а для использования к имени ресурса нужно добавить специальный признак "::".

```
//--- использование ресурса в самом скрипте
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"::Files\\triangle.bmp");
```

Чтобы использовать этот же ресурс из другой программы, например, эксперта, нужно к имени ресурса дополнительно добавить путь к EX5-файлу относительно папки каталог\_данных\_терминала\MQL5\ и имя EX5-файла этого скрипта - Draw\_Triangles\_Script.ex5. Пусть скрипт лежит в стандартной папке каталог\_данных\_терминала\MQL5\Scripts\, тогда вызов нужно написать таким образом:

```
//--- использование ресурса скрипта в эксперте
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"\\Scripts\\Draw_Triangles_Script.e
```

Если при обращении к ресурсу в другом EX5-файле не указать путь к этому исполняемому файлу, то исполняемый файл ищется в той же папке, где находится и обратившаяся за ресурсом программа. Это означает, что если в советнике запрашивается ресурс из файла Draw\_Triangles\_Script.ex5 без указания пути, например, так:

```
//--- запрос ресурса скрипта в эксперте без указания пути
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"Draw_Triangles_Script.ex5::Files\\
```

то файл будет искааться в папке каталог\_данных\_терминала\MQL5\Experts\, если сам советник находится в папке каталог\_данных\_терминала\MQL5\Experts\.

## Работа с пользовательскими индикаторами, подключенными в качестве ресурсов

Для работы mq5-программ может потребоваться один или несколько пользовательских индикаторов, все они могут быть включены в код исполняемой mq5-программы. Включение индикаторов в качестве ресурсов позволяет упростить распространение программ.

Пример подключения и использования пользовательского индикатора SampleIndicator.ex5, расположенного в папке: каталог\_данных\_терминала\MQL5\Indicators\:

```
//+-----+
//|                               SampleEA.mq5  |
//|                               Copyright 2013, MetaQuotes Software Corp.  |
//|                               https://www.mql5.com  |
//+-----+
#resource "\\Indicators\\SampleIndicator.ex5"
int handle_ind;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//---
    handle_ind=iCustom(_Symbol,_Period,"::Indicators\\SampleIndicator.ex5");
```

```

if(handle__ind==INVALID_HANDLE)
{
    Print("Expert: iCustom call: Error code=",GetLastError());
    return(INIT_FAILED);
}
//---
return(INIT_SUCCEEDED);
}

```

Случай, когда пользовательский индикатор в функции [OnInit\(\)](#) создает одну или несколько копий себя, требует отдельного рассмотрения. Напомним, что для использования ресурса из mq5-программы его необходимо указывать в виде: <путь\_имя\_файла\_EX5>::<имя\_ресурса>.

Например, если индикатор SampleIndicator.ex5 включается в советник SampleEA.ex5 в качестве ресурса, то путь к самому себе, указанный при вызове [iCustom\(\)](#) в функции инициализации пользовательского индикатора, будет выглядеть следующим образом: "\Experts\SampleEA.ex5::Indicators\SampleIndicator.ex5". При явном указании данного пути пользовательский индикатор SampleIndicator.ex5 будет жестко привязан к советнику SampleEA.ex5 и теряет способность самостоятельной работы.

Путь до самого себя можно получить при помощи функции [GetRelativeProgramPath\(\)](#), пример использования которой приведен ниже:

```

//+-----+
//|                                                 SampleIndicator.mq5 |
//|                                                 Copyright 2013, MetaQuotes Software Corp. |
//|                                                 https://www.mql5.com |
//+-----+
#property indicator_separate_window
#property indicator_plots 0
int handle;
//+-----+
//| Custom indicator initialization function      |
//+-----+
int OnInit()
{
//--- неправильный способ указания ссылки на себя
//--- string path="\Experts\SampleEA.ex5::Indicators\SampleIndicator.ex5";
//--- правильный способ получения ссылки на себя
    string path=GetRelativeProgramPath();
//--- indicator buffers mapping
    handle=iCustom(_Symbol,_Period,path,0,0);
    if(handle==INVALID_HANDLE)
    {
        Print("Indicator: iCustom call: Error code=",GetLastError());
        return(INIT_FAILED);
    }
    else Print("Indicator handle=",handle);
}
//---
return(INIT_SUCCEEDED);

```

```

    }

//+-----+
//| GetRelativeProgramPath
//+-----+

string GetRelativeProgramPath()
{
    int pos2;
    //--- получаем абсолютный путь к программе
    string path=MQLInfoString(MQL_PROGRAM_PATH);
    //--- находим позицию подстроки "\MQL5\
    int pos=StringFind(path, "\\MQL5\\");
    //--- подстрока не найдена - ошибка
    if(pos<0)
        return(NULL);
    //--- пропускаем каталог "\MQL5"
    pos+=5;
    //--- пропускаем лишние \
    while(StringGetCharacter(path, pos+1)=='\\')
        pos++;
    //--- если это ресурс, возвращаем путь относительно MQL5-каталога
    if(StringFind(path, ":::", pos)>=0)
        return(StringSubstr(path, pos));
    //--- найдем разделитель для первого подкаталога в MQL5 (например, MQL5\Indicators)
    //--- если его нет, то вернем путь относительно MQL5-каталога
    if((pos2=StringFind(path, "\\\", pos+1))<0)
        return(StringSubstr(path, pos));
    //--- вернем путь относительно подкаталога (например, MQL5\Indicators)
    return(StringSubstr(path, pos2+1));
}

//+-----+
//| Custom indicator iteration function
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double& price[])
{
    //--- return value of prev_calculated for next call
    return(rates_total);
}

```

## Ресурсные переменные

Ресурсы можно объявлять с помощью ресурсных переменных и обращаться с ними так, как будто они являются переменной соответствующего типа. Формат объявления:

```
#resource путь_к_файлу_ресурса as тип_ресурсной_переменной имя_ресурсной_переменной
```

Примеры объявлений:

```
#resource "data.bin" as int ExtData[]          // объявление массива числового типа
#resource "data.bin" as MqlRates ExtData[]      // объявление массива простых структур
//--- строки
#resource "data.txt" as string ExtCode         // объявление строки, содержащей данные
//--- графические ресурсы
#resource "image.bmp" as bitmap ExtBitmap[]     // объявление одномерного массива, состоящего из изображений
#resource "image.bmp" as bitmap ExtBitmap2[][]   // объявление двумерного массива, состоящего из изображений
```

При таком объявлении к данным ресурса можно адресоваться только через переменную, автоматическая адресация через "`::<resource name>`" не работает.

```
#resource "\\Images\\euro.bmp" as bitmap euro[][]

#resource "\\Images\\dollar.bmp"

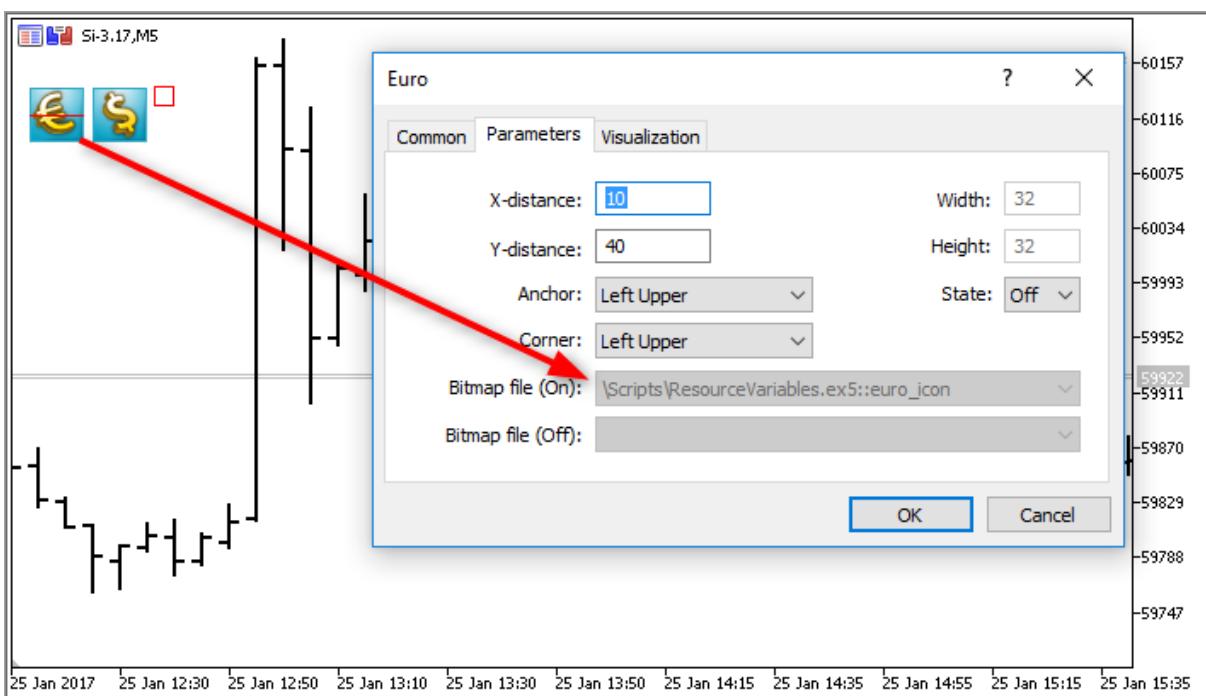
//+-----+
//|  Функция создания объекта OBJ_BITMAP_LABEL с помощью ресурса |+
//+-----+

void Image(string name, string rc, int x, int y)
{
    ObjectCreate(0, name, OBJ_BITMAP_LABEL, 0, 0, 0);
    ObjectSetInteger(0, name, OBJPROP_XDISTANCE, x);
    ObjectSetInteger(0, name, OBJPROP_YDISTANCE, y);
    ObjectSetString(0, name, OBJPROP_BMPFILE, rc);
}

//+-----+
//| Script program start function |+
//+-----+

void OnStart()
{
    //--- выведем размеры картинки [width, height], которая хранится в ресурсной переменной
    Print(ArrayRange(euro,1), ", ", ArrayRange(euro,0));
    //--- изменим картинку в euro - нарисуем красную горизонтальную полосу посередине
    for(int x=0;x<ArrayRange(euro,1);x++)
        euro[ArrayRange(euro,1)/2][x]=0xFFFF0000;
    //--- создадим графический ресурс с помощью ресурсной переменной
    ResourceCreate("euro_icon", euro, ArrayRange(euro,0), ArrayRange(euro,1), 0, 0, ArrayRange(euro,1));
    //--- создадим объект графическая метка Euro, которому выставим картинку из ресурса euro
    Image("Euro", "::euro_icon", 10, 40);
    //--- другой способ использования ресурса, рисовать в него мы не можем
    Image("USD", "::Images\\dollar.bmp", 15+ArrayRange(euro,1), 40);
    //--- прямой способ адресации к ресурсу euro.bmp недоступен, так как он уже объявлен в скрипте
    Image("E2", "::Images\\euro.bmp", 20+ArrayRange(euro,1)*2, 40); // произойдет ошибка
}
```

Результат выполнения скрипта - созданы только два объекта `OBJ_BITMAP_LABEL` из трех. При этом на изображении первого объекта мы видим красную полоску посередине.



Важным преимуществом использования ресурсов является то, что ресурсные файлы перед включением в исполняемый EX5-файл перед компиляцией автоматически сжимаются. Таким образом, использование ресурсных переменных позволяет не только упаковывать необходимые для работы данные прямо в исполняемый EX5-файл, но и сокращает количество и общий размер файлов по сравнению с обычным способом написания mq5-программ.

Использование ресурсных переменных особенно удобно для публикации продуктов в [Маркете](#).

## Особенности

- Специальный тип ресурсной переменной *bitmap* указывает компилятору, что ресурс является графическим изображением. Такие переменные получают тип *uint*.
- Ресурсная переменная-массив типа *bitmap* может иметь две размерности, в этом случае размер массива будет установлен как [высота\_картинки][ширина\_картинки]. В случае если указан массив одной размерности, то количество элементов будет равно произведению высота\_картинки\*ширина\_картинки.
- При загрузке 24 битного изображения для всех пикселей изображения компонента альфа-канала устанавливается в значение 255.
- При загрузке 32 битного изображения без альфа-канала, также, для всех пикселей изображения компонента альфаканала устанавливается в значение 255.
- При загрузке 32 битного изображения с альфа-каналом никаких манипуляций с пикселями не происходит.
- Размер файла ресурса не может быть больше 128 Mb.
- Для строковых файлов производится автоматическое определение кодировки по наличию BOM (заголовку). Если BOM отсутствует, то кодировка определяется по содержимому файла. Поддерживаются файлы в кодировке ANSI, UTF-8 и UTF-16. При чтении данных из файлов все строки переводятся в Unicode.

## Программы на OpenCL

Использование ресурсных строковых переменных может существенно облегчить написание некоторых программ. Например, вы можете написать код [OpenCL-программы](#) в отдельном CL-файле, а затем включить этот файл в виде строки в ресурсы вашей MQL5-программы.

```
#resource "seascape.cl" as string cl_program
...
int context;
if((cl_program=CLProgramCreate(context,cl_program)!=INVALID_HANDLE)
{
    //--- выполняем дальнейшие действия с OpenCL программой
}
```

В данном примере без использования ресурсной переменной *cl\_program* вам пришлось бы описывать весь код в виде одной большой строковой переменной.

#### Смотри также

[ResourceCreate\(\)](#), [ResourceSave\(\)](#), [PlaySound\(\)](#), [ObjectSetInteger\(\)](#), [ChartApplyTemplate\(\)](#),  
[Файловые операции](#)

## ВЫЗОВ ИМПОРТИРУЕМЫХ ФУНКЦИЙ

Для импорта функций во время выполнения mql5-программы используется раннее связывание. Это значит, что если в программе есть вызов импортируемой функции, то соответствующий модуль (ex5 или dll) загружается в процессе загрузки программы. Библиотеки MQL5 и DLL выполняются в потоке вызывающего модуля.

Не рекомендуется использовать полностью квалифицированное имя загружаемого модуля вида *Drive:\Directory\FileName.Ext*. Библиотеки MQL5 загружаются из папки *terminal\_dir\MQL5\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 из которой был запущен модуль, импортирующий dll. Под модулем понимается эксперт, скрипт, индикатор или библиотека EX5;
2. Директория каталог\_данных\_терминала\MQL5\Libraries ([TERMINAL\\_DATA\\_PATH](#)\MQL5\Libraries);
3. Директория, из которой был запущен клиентский терминал MetaTrader 5;
4. Системная директория;
5. Директория Windows;
6. Текущая директория;
7. Директории, перечисленные в системной переменной PATH.

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

Перед загрузкой эксперта (скрипта, индикатора) формируется общий список всех библиотечных модулей EX5, которые предполагается использовать как из загружаемого эксперта (скрипта, индикатора), так и из библиотек из этого списка. Таким образом обеспечивается однократная загрузка многократно используемых библиотечных модулей EX5. Библиотеки пользуются [предопределенными переменными](#) вызвавшего их эксперта (скрипта, индикатора).

Поиск импортируемой библиотеки EX5 производится в следующей последовательности:

1. Директория, путь к которой задается относительно директории импортирующего EX5 эксперта (скрипта, индикатора);
2. Директория каталог\_терминала\MQL5\Libraries;
3. Директория MQL5\Libraries в общей директории всех клиентских терминалов MetaTrader 5 (Common\MQL5\Libraries).

Функции, [импортируемые](#) из DLL в mql5-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 mq5-программа) должна "видеть" прототип вызываемой (импортируемой из DLL) функции, для того чтобы правильно сложить параметры на стек;
- вызывающая функция (в нашем случае mq5-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 можно использовать параметры со значениями по умолчанию.

В случае если соответствующая библиотека не смогла загрузиться, либо установлен запрет на использование DLL, либо импортируемая функция не была найдена - 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 OK.

## Передача параметров

Все параметры [простых типов](#) передаются по значению, если явно не указано, что они передаются по ссылке. При передаче [строки](#) передается адрес буфера скопированной строки; если строка передается по ссылке, то в функцию, импортируемую из DLL, передается адрес буфера именно этой строки без копирования.

[Структуры](#), содержащие динамические массивы, строки, классы, другие сложные структуры, а также статические либо [динамические массивы](#) перечисленных объектов, не могут быть переданы в качестве параметра в импортируемую функцию.

При передаче в DLL массива всегда (независимо от флага [AS\\_SERIES](#)) передается адрес начала буфера данных. Функция внутри DLL ничего не знает о флаге AS\_SERIES, переданный массив является статическим массивом неизвестной длины, для указания размера массива используйте дополнительный параметр.

## Ошибки выполнения

В исполняющей подсистеме клиентского терминала существует возможность сохранения [кода ошибки](#) в случае ее возникновения при выполнении mql5-программы. Для каждой исполняемой mql5-программы предусмотрена предопределенная переменная [\\_LastError](#).

Перед запуском функции [OnInit](#) переменная `_LastError` обнуляется. При возникновении ошибочной ситуации во время вычислений или в процессе вызова встроенной функции переменная `_LastError` принимает соответствующий код ошибки. Значение, сохраненное в этой переменной, можно получить при помощи функции [GetLastError\(\)](#).

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

- деление на ноль;
- выход за пределы массива;
- использование некорректного [указателя объекта](#);

## Тестирование торговых стратегий

Идея автоматической торговли привлекательна тем, что торговый робот может без устали работать 24 часа в сутки и семь дней в неделю. Робот не знает усталости, сомнений и страха, ему не ведомы психологические проблемы. Достаточно четко формализовать торговые правила и реализовать их в виде алгоритмов, и робот готов неустанно трудиться. Но прежде необходимо убедиться в соблюдении двух важных условий:

- эксперт совершает [торговые операции](#) в соответствии с правилами торговой системы;
- торговая стратегия, реализованная в эксперте, показывает прибыль на истории.

Для получения ответов на эти вопросы предназначен [тестер стратегий](#), входящий в состав клиентского терминала MetaTrader 5.

В данном разделе рассмотрены все особенности тестирования и оптимизации программ в тестере стратегий:

- [Ограничения работы функций в тестере торговых стратегий](#)
- [Режимы генерации тиков](#)
- [Моделирование спреда](#)
- [Использование реальных тиков при тестировании](#)
- [Глобальные переменные клиентского терминала](#)
- [Расчет индикаторов при тестировании](#)
- [Загрузка истории при тестировании](#)
- [Мультивалютное тестирование](#)
- [Моделирование времени в тестере](#)
- [Графические объекты при тестировании](#)
- [Функция OnTimer\(\) в тестере](#)
- [Функция Sleep\(\) в тестере](#)
- [Использование тестера для задач оптимизации в математических вычислениях](#)
- [Синхронизация баров при тестировании в режиме "Только цены открытия"](#)
- [Функция IndicatorRelease\(\) в тестере](#)
- [Обработка событий в тестере](#)
- [Агенты тестирования](#)
- [Обмен данными между терминалом и агентом](#)
- [Использование общей папки всех клиентских терминалов](#)
- [Использование DLL](#)

## Ограничения работы функций в тестере торговых стратегий

Существуют ограничения работы некоторых функций в тестере стратегий клиентского терминала.

### Функции Comment(), Print() и PrintFormat()

Для увеличения быстродействия при оптимизации параметров советника функции [Comment\(\)](#), [Print\(\)](#) и [PrintFormat\(\)](#) не выполняются. Исключением является использование этих функций внутри обработчика [OnInit\(\)](#). Это позволяет облегчить поиск причин ошибок при их возникновении.

## Функции Alert(), MessageBox(), PlaySound(), SendFTP, SendMail(), SendNotification(), WebRequest()

Функции взаимодействия с "внешним миром" [Alert\(\)](#), [MessageBox\(\)](#), [PlaySound\(\)](#), [SendFTP\(\)](#), [SendMail\(\)](#), [SendNotification\(\)](#) и [WebRequest\(\)](#) в тестере стратегий не выполняются.

## Режимы генерации тиков

Эксперт на языке MQL5 представляет из себя программу, которая запускается каждый раз в ответ на некое внешнее воздействие - [событие](#). Для каждого предопределенного события в эксперте есть соответствующая этому событию функция - [обработчик события](#).

Основным событием для эксперта является изменение цены - [NewTick](#), и поэтому для тестирования экспертов необходимо генерировать тиковые последовательности. В тестере клиентского терминала MetaTrader 5 реализовано 3 режима генерации тиков:

- Все тики
- Цены OHLC с минутных баров (1 Minute OHLC)
- Только цены открытия

Базовым и наиболее детальным режимом генерации является режим "Все тики", остальные два режима являются упрощением основного и будут описаны в сравнении с режимом "Все тики". Рассмотрим все три режима, чтобы понять в чем различие между ними.

### Все тики

История котировок по финансовым инструментам передается от торгового сервера в клиентский терминал MetaTrader 5 в виде экономно упакованных блоков минутных баров. Подробную информацию о том, как происходит запроса и построение требуемых таймфреймов можно получить из раздела справки [Организация доступа к данным](#).

Минимальным элементом ценовой истории является минутный бар, из которого можно получить информацию о четырех значениях цены:

- Open - цена, по которой открылся минутный бар;
- High - максимум, который достигался в течение этого минутного бара;
- Low - минимум, который достигался в течение этого минутного бара;
- Close - цена закрытия бара.

Новый минутный бар открывается не в тот момент, когда начинается новая минута (количество секунд становится равным 0), а когда приходит тик - изменение цены хотя бы на один пункт. На рисунке показан первый минутный бар новой торговой недели, который имеет время открытия 2011.01.10. 00:00. Ценовой разрыв между пятницей и понедельником, который мы видим на

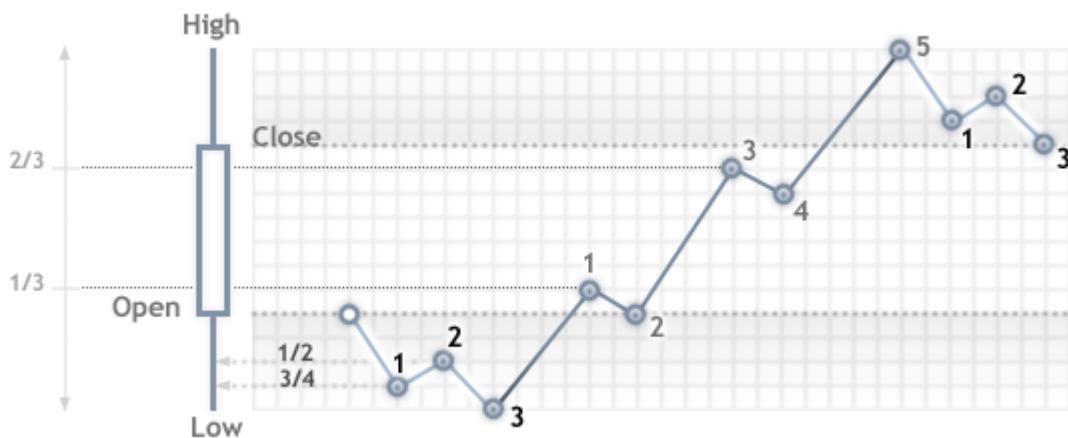
графике, является обычным явлением, так как даже в выходные дни валютные курсы изменяются в ответ на поступающие новости.



Для этого бара нам только известно, что данный минутный бар открыт 10 января 2011 года в 00 часов 00 минут, но ничего неизвестно о секундах. Это могло быть время 00:00:12 или 00:00:36 (12 или 36 секунд после начала нового дня) или любое другое время в пределах этой минуты. Но мы знаем точно, что в момент открытия нового минутного бара цена Open по EURUSD была на уровне 1.28940.

Точно также мы не знаем с точностью до секунды, когда пришел тик, соответствующий цене закрытия рассматриваемого минутного бара, известно только одно - это последняя цена на минутном баре, которая и была записана как цена Close. Для данной минуты это оказалась цена 1.28958. Время появления цен High и Low также неизвестно, но мы знаем, что максимальная и минимальная цена точно побывала на уровнях 1.28958 и 1.28940 соответственно.

Для тестирования торговой стратегии нам необходима тиковая последовательность, на которой будет эмулироваться работа эксперта. Таким образом, для каждого минутного бара нам известны **4 контрольные точки**, о которых мы точно можем сказать, что цена там побывала. Если бар имеет только 4 тика, то для тестирования этой информации достаточно, но обычно тиковый объем больше 4. Значит, необходимо сгенерировать дополнительные контрольные точки для тиков, которые приходили между ценами Open, High, Low и Close. Принцип генерации тиков в режиме "Все тики" описан в статье [Алгоритм генерации тиков в тестере стратегий терминала MetaTrader 5](#), рисунок из которой представлен ниже.



При тестировании в режиме "Все тики" функция [OnTick\(\)](#) эксперта будет вызываться на каждой контрольной точке, каждая контрольная точка - это тик из сгенерированной последовательности. Эксперт будет получать время и цену смоделированного тика так же, как и при работе в онлайне.

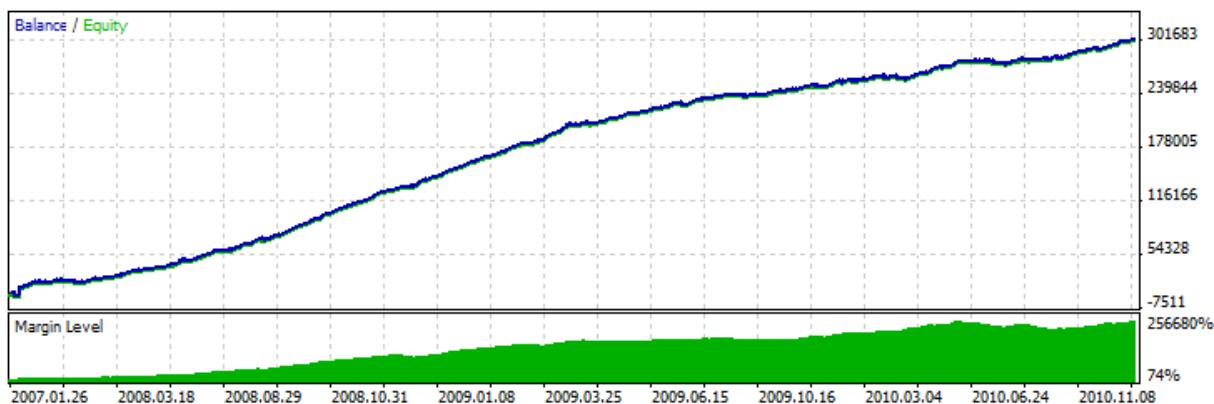
**Важно:** режим тестирования "Все тики" является самым точным, но при этом и самым затратным по времени. Для первичной оценки большинства торговых стратегий обычно достаточно использовать один из двух других режимов тестирования.

## 1 minute OHLC

Тестирование в режиме "Все тики" является самым точным из трех режимов, но в то же время и самым медленным. Запуск обработчика `OnTick()` происходит на каждом тике, а тиковый объем может быть достаточно большим. Для стратегий, которым не важно, в какой тиковой последовательности развивалась цена в течение бара, существует более быстрый и более грубый режим моделирования - "1 minute OHLC".

В режиме "1 minute OHLC" тиковая последовательность строится только по OHLC ценам минутных баров, количество сгенерированных контрольных точек существенно уменьшается - следовательно, уменьшается время тестирования. Запуск функции `OnTick()` производится на всех контрольных точках, которые строятся по ценам OHLC минутных баров.

Отказ от генерации дополнительных промежуточных тиков между ценами Open, High, Low и Close приводит к появлению жесткой детерминированности в развитии цены с того момента, как определена цена Open. Это дает возможность для создания "Грааля тестирования", который показывает красивый восходящий график баланса при тестировании. Пример такого Грааля представлен в CodeBase - [Grr-al](#).



На рисунке представлен очень привлекательный график тестирования этого эксперта. Как он получен? Для минутного бара известно 4 цены, и для них точно известно, что первой идет цена Open, а последней идет цена Close. Между ними есть цены High и Low, последовательность их наступления неизвестна, но известно, что цена High больше или равна цене Open (цена Low меньше или равна цене Open).

Достаточно определить момент поступления цены Open и затем анализировать следующий тик, чтобы определить что перед нами - High или Low. Если цена ниже цены Open, значит, перед нами цена Low - покупаем на этом тике, следующий тик будет соответствовать цене High, на котором закрываем покупку и открываем продажу. Следующий тик последний, это цена Close, на нем закрываем продажу.

Если после цены пришел тик с ценой больше цены открытия, то последовательность сделок обратная. Отработаем в таком мошенническом режиме минутный бар и ждем следующий. При тестировании такого эксперта на истории все идет хорошо, но стоит запустить его в онлайне, и сказка рассыпается - линия баланса по-прежнему ровная, но идет вниз. Для быстрого разоблачения трюка достаточно прогнать такой советник в режиме "Все тики".

**Важно:** если результаты тестирования эксперта на грубых режимах тестирования ("1 minute OHLC" и "Только цены открытия") слишком хороши, обязательно протестируйте его в режиме "Все тики".

## Только цены открытия

В данном режиме происходит генерация тиков по ценам OHLC таймфрейма, выбранного для тестирования. При этом функция эксперта OnTick() запускается только в начале бара по цене Open. Из-за этой особенности стоп-уровни и отложенные ордера могут срабатывать не по заявленной цене (особенно при тестировании на старших таймфреймах). В обмен за это мы получаем возможность быстро провести оценочное тестирование эксперта.

Исключением при генерации тиков в режиме "Только цены открытия" являются периоды W1 и MN1: для этих таймфреймов тики генерируются для цен OHLC каждого дня, а не для цен OHLC недели или месяца соответственно.

Например, производится тестирование советника на EURUSD H1 в режиме "Только цены открытия". В этом случае общее количество тиков (контрольных точек) будет не больше 4\*количество часовых баров, попавших в тестируемый интервал. Но при этом вызов обработчика OnTick() производится только на открытии часового бара. На остальных ("скрытых" от эксперта) тиках происходят проверки, необходимые для корректного тестирования:

- вычисление маржевых требований;
- срабатывание Stop Loss и Take Profit;
- срабатывание отложенных ордеров;
- удаление отложенных ордеров с истекшим временем.

Если нет открытых позиций или отложенных ордеров, то необходимости в данных проверках на скрытых тиках нет и прирост скорости может оказаться существенным. Данный режим "Только цены открытия" хорошо подходит для тестирования стратегий, которые совершают сделки только на открытии бара и не используют отложенные ордера, а также не используют ордера StopLoss,, TakeProfit. Для класса таких стратегий сохраняется вся необходимая точность тестирования.

В качестве примера эксперта, которому не важно в каком режиме тестироваться, приведем советник Moving Average из стандартной поставки. Логика этого эксперта построена таким образом, что все решения принимаются на открытии бара и сделки проводятся сразу же без использования отложенных ордеров. Запустим тестирование эксперта на EURUSD H1 на интервале с 2010.09.01 по 2010.12.31 и сравним графики. На рисунке показаны графики баланса из отчета тестера для всех трех режимов.



Как видите, графики на разных режимах тестирования абсолютно одинаковы для советника Moving Average из стандартной поставки.

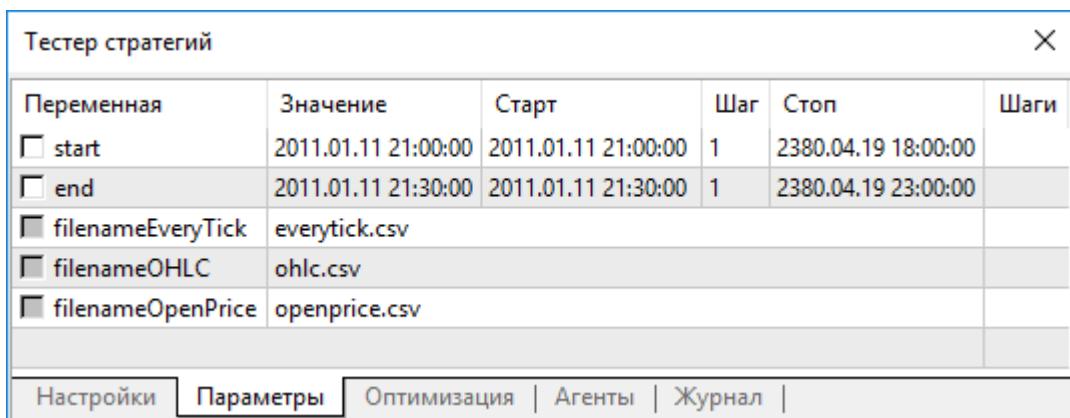
Существует ряд ограничений применения режима "Только цены открытия":

- Нельзя использовать [режим торговли "Произвольная задержка"](#);
- В тестируемом эксперте невозможно обратиться к данным более низкого [таймфрейма](#), чем тот, что используется для тестирования/оптимизации. Например, если тестирование/оптимизация осуществляется на периоде H1, то вы можете обращаться к данным H2, H3, H4 и т.д., но не к данным M30, M20, M10 и т.д. Помимо этого, более старшие таймфреймы, к которым идет обращение, должны быть кратными таймфрейму тестирования. Например, при тестировании на периоде M20 нельзя обратиться к таймфрейму M30, но можно к H1. Эти ограничения обусловлены невозможностью получить данные более низких и не кратных таймфреймов из баров, генерируемых при тестировании/оптимизации.

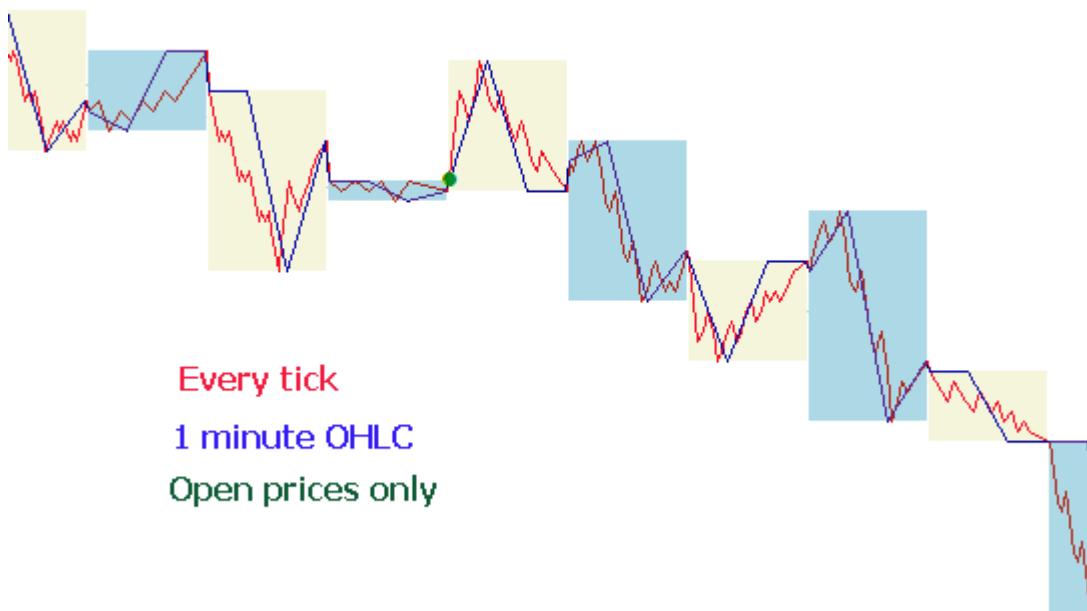
- Ограничения по обращению к данным других таймфреймов распространяются и на другие символы, чьи данные используются советником. Однако в этом случае ограничением для каждого символа служит первый таймфрейм, к которому произошло обращение во время тестирования/оптимизации. Например, тестирование осуществляется на символе и периоде EURUSD H1, советник в первый раз обратился к символу GBPUSD M20. В этой ситуации советник в дальнейшем может использовать данные EURUSD H1, H2, и т.д., а также GBPUSD M20, H1, H2 и т.д.

**Важно:** режим "Только цены открытия" является самым быстрым по времени тестирования, но подходит не для всех торговых стратегий. Выбирайте необходимый режим тестирования исходя из особенностей работы торговой системы.

В завершение раздела о режимах моделирования приведем визуальное сравнение разных режимов генерации тиков для EURUSD для двух баров M15 на интервале с 2011.01.11 21:00:00 - 2011.01.11 21:30:00. Запись тиков произведена с помощью эксперта WriteTicksFromTester.mq5 в разные файлы, окончание имен этих файлов задаются [input-параметрах](#) filenameEveryTick, filenameOHLC и filenameOpenPrice.



Чтобы получить три файла с тремя тиковыми последовательностями (для каждого из режимов "Все тики", "OHLC на минутных барах" и "Только цены открытия") советник был запущен трижды в соответствующих режимах на одиночных прогонах. Затем данные из этих трех файлов с помощью индикатора TicksFromTester.mq5 были выведены на график. Код индикатора прилагается к статье.



По умолчанию все [файловые операции](#) в языке MQL5 производятся в пределах "файловой песочницы" и при тестировании эксперту доступна только собственная "файловая песочница". Для того чтобы индикатор и эксперт при тестировании работали с файлами из одной папки, использовался [флаг FILE\\_COMMON](#). Пример кода из эксперта:

```
//--- откроем файл
file=FileOpen(filename,FILE_WRITE|FILE_CSV|FILE_COMMON,";");
//--- проверим успешность операции
if(file==INVALID_HANDLE)
{
    PrintFormat("Не удалось открыть на запись файл %s. Код ошибки=%d",filename,GetLastError());
    return;
}
else
{
    //--- сообщим о записи в общую папку всех клиентских терминалов и ее местоположение
    PrintFormat("Файл будет записан в папке %s",TerminalInfoString(TERMINAL_COMMONDIR));
}
```

В индикаторе для чтения данных также использовался [флаг FILE\\_COMMON](#), это позволило избежать переноса необходимых файлов вручную из одной папки в другую.

```
//--- откроем файл
int file=FileOpen(fname,FILE_READ|FILE_CSV|FILE_COMMON,";");
//--- проверим успешность операции
if(file==INVALID_HANDLE)
{
    PrintFormat("Не удалось открыть на чтение файл %s. Код ошибки=%d",fname,GetLastError());
    return;
}
else
```

```
{
//--- сообщим местонахождение общей папки всех клиентских терминалов
PrintFormat("Файл будет прочитан из папки %s", TerminalInfoString(TERMINAL_COMMON));
}
```

## Моделирование спреда

Разница между ценами Bid и Ask называется спредом. При тестировании спред не моделируется, а берется из исторических данных. Если в исторических данных спред меньше или равен нулю, то используется последний известный на момент генерации спред.

В тестере спред всегда считается плавающим. То есть [SymbolInfoInteger\(symbol, SYMBOL\\_SPREAD\\_FLOAT\)](#) всегда возвращает true.

Кроме того, в исторических данных хранятся значения тиковых и торговых объемов. Для хранения и получения данных используется специальная структура [MqlRates](#):

```
struct MqlRates
{
    datetime time;           // время открытия бара
    double open;             // цена открытия Open
    double high;             // наивысшая цена High
    double low;              // наименьшая цена Low
    double close;            // цена закрытия Close
    long tick_volume;        // тиковый объем
    int spread;              // спред
    long real_volume;        // биржевой объем
};
```

## Использование реальных тиков при тестировании

Тестирование и оптимизация на реальных тиках являются максимально приближенными к реальным условиям. Вместо сгенерированных на основе минутных данных используются реальные тики, накопленные по финансовым инструментам брокером. Это – тики с бирж и от поставщиков ликвидности.

Чтобы обеспечить наибольшую точность при тестировании, в режиме реальных тиков также используются и минутные бары. По ним проверяются и корректируются тиковые данные. Это также позволяет избежать расхождения графиков в тестере и клиентском терминале.

Тестер проверяет соответствие тиковых данных параметрам минутного бара: тик не должен выходить за пределы цен High/Low бара, открывающий и закрывающий минуту тик должен совпадать с ценами Open/Close бара. Также сравнивается объем. При выявлении несовпадения отбрасываются все тики, соответствующие этому минутному бару. Вместо них будут использованы сгенерированные тики (как в режиме "Все тики").

Если в истории символа есть минутный бар, но тиковых данных за эту минуту нет, тестер генерирует тики в режиме "Все тики". Это позволяет выстроить правильный график в тестере в случае неполных тиковых данных у брокера.

Если в истории символа нет минутного бара, но тиковые данные за эту минуту есть, они могут быть использованы в тестере. Например, бары биржевых символов формируются по ценам Last. Если с сервера приходят только тики с ценами Bid/Ask без цены Last, бар не будет сформирован. Тестер будет использовать эти тиковые данные, поскольку они не противоречат минутным.

Тиковые данные могут не совпадать с минутными барами по различным причинам. Например, из-за обрывов связи или иных сбоев при передаче данных от источника в клиентский терминал. При тестировании минутные данные считаются более достоверными.

При тестировании на реальных тиках учитывайте следующие особенности:

- При запуске тестирования синхронизируются не только тиковые, но и минутные данные по инструменту.
- Тики хранятся в кэше символа в тестере стратегий. Размер кэша – не более 128 000 тиков. При поступлении новых тиков самые старые данные из него выталкиваются. Однако при помощи функции [CopyTicks](#) можно получить тики и за пределами кэша (только при тестировании по реальным тикам). В этом случае данные будут запрошены из базы тиков тестера, которая полностью соответствует аналогичной базе клиентского терминала. В эту базу никакие корректировки по минутным барам не вносятся. Поэтому тики в ней могут отличаться от тиков, находящихся в кэше.

## Глобальные переменные клиентского терминала

При тестировании [глобальные переменные клиентского терминала](#) также эмулируются, но они никак не связаны с настоящими [глобальными переменными терминала](#), которые можно увидеть в терминале по кнопке F3. Это означает, что все операции с глобальными переменными терминала при тестировании производятся вне самого клиентского терминала (в агенте тестирования).

## Расчет индикаторов при тестировании

В режиме реального времени значения индикаторов вычисляются на каждом тике. В тестере принята экономичная модель вычисления индикаторов - [индикаторы пересчитываются](#) только непосредственно перед тем, как запускается на исполнение эксперт. Это означает, что пересчет значений индикаторов производится перед вызовом функций OnTick(), OnTrade() и OnTimer().

Неважно, есть вызов индикатора в конкретном обработчике события или нет, все индикаторы, чьи хэндлы были созданы функцией [iCustom\(\)](#) или [IndicatorCreate\(\)](#), будут принудительно пересчитаны перед вызовом функции-обработчика события.

Следовательно, при тестировании в режиме "Все тики" расчет индикаторов происходит перед каждым вызовом [OnTick\(\)](#). Если в эксперте с помощью функции [EventSetTimer\(\)](#) включен таймер, то индикаторы будут пересчитаны перед каждым вызовом обработчика [OnTimer\(\)](#). Следовательно, время тестирования может многократно возрасти при использовании в эксперте индикатора, написанного неоптимальным образом.

## Загрузка истории при тестировании

История по тестируемому инструменту синхронизируется и закачивается терминалом с торгового сервера перед запуском процесса тестирования. При этом в первый раз терминал скачивает с

торгового сервера сразу всю доступную по тестируемому инструменту историю, чтобы впоследствии не обращаться за ней. В дальнейшем происходит лишь докачка новых данных.

Агент тестирования получает от клиентского терминала историю по тестируемому инструменту сразу же после запуска тестирования. Если в процессе тестирования используются данные по другим инструментам (например, это мультивалютный эксперт), то в этом случае агент тестирования запрашивает у клиентского терминала требуемую историю при первом же обращении. Если исторические данные имеются на терминале, они сразу передаются на агенты тестирования. Если данные отсутствуют, терминал запросит и скачает их с сервера, а затем передаст на агенты тестирования.

Обращение к дополнительным инструментам происходит и в том случае, когда вычисляется цена кросс-курса при торговых операциях. Например, при тестировании стратегии на EURCHF с валютой депозита в долларах США перед обработкой первой же торговой операцией агент тестирования запрашивает у клиентского терминала историю по EURUSD и USDCHF, хотя в стратегии нет прямого обращения к этим инструментам.

Перед началом тестирования мультивалютной стратегии рекомендуется предварительно скачать все необходимые исторические данные на клиентском терминале. Это позволит избежать задержек при тестировании/оптимизации, связанных с докачкой данных. Закачать историю можно, например, путем открытия соответствующих графиков и прокрутки их к началу истории. Пример принудительной загрузки истории в торговый терминал приведен в документации по MQL5 в разделе [Организация доступа к данным](#).

Тестерные агенты в свою очередь получают историю от терминала и также в упакованном виде. При повторном тестировании загрузка тестером истории из терминала уже не происходит, потому что данные есть от предыдущего запуска тестера.

- Терминал загружает историю с торгового сервера только один раз при первом обращении агента к терминалу за историей для тестируемого символа. История загружается в упакованном виде, чтобы сократить трафик.
- Тики не пересыпаются по сети, они генерируются на тестерных агентах.

## Мультивалютное тестирование

Тестер позволяет проводить проверку на истории стратегий, торгующих на нескольких инструментах. Такие эксперты условно называют мультивалютными, так как изначально в предыдущих платформах тестирование проводилось только для одного инструмента. В тестере же терминала MetaTrader 5 можно моделировать торговлю по всем доступным инструментам.

История по используемым инструментам закачивается тестером из **клиентского терминала** (не с торгового сервера!) автоматически при первом обращении к данному инструменту.

Агент тестирования закачивает только недостающую историю с небольшим запасом, чтобы обеспечить необходимые данные на истории для расчета индикаторов на момент начала тестирования. Минимальный объем истории при скачивании с торгового сервера для таймфреймов D1 и меньше составляет один год. Таким образом, если запускается тестирование на интервале 2010.11.01-2010.12.01 (тестирование на интервале в один месяц) с периодом M15 (каждый бар равен 15 минутам), то у терминала будет запрошена история по инструменту за весь 2010 год. Для таймфреймов Weekly будет запрошена история в 100 баров, что составляет примерно два года (в

году 52 недели). Для тестирования на месячном таймфрейме Monthly агент запросит историю за 8 лет (12 месяцев \* 8 лет = 96 месяцев).

Если не удается по каким-либо причинам обеспечить необходимое количество баров перед началом тестирования, то дата начала будет автоматически сдвинута от прошлого к настоящему для того чтобы обеспечить такое количество баров.

При тестировании эмулируется также и "[Обзор рынка](#)", из которого можно получать [информацию по инструментам](#). По умолчанию в начале тестирования в "Обзоре рынка" тестера есть только один символ - символ на котором запущено тестирование. Все необходимые символы подключаются к "Обзору рынка" тестера (не терминала!) автоматически при обращении к ним.

Перед началом тестирования мультивалютного эксперта необходимо выбрать требуемые для тестирования инструменты в "Обзоре рынка" терминала и [подкачать данные](#) на нужную глубину. При первом же обращении к "чужому" символу будет автоматически произведена синхронизация по этому символу между агентом тестирования и клиентским терминалом. "Чужой" символ - это символ, отличающийся от того, на котором запущено тестирование.

Обращение к данным чужого символа происходят в следующих случаях:

- использование [функций технических индикаторов](#) и [IndicatorCreate\(\)](#) на паре символ/таймфрейм;
- запрос к "Обзору рынка" (Market Watch) по чужому символу:
  1. [SeriesInfoInteger](#)
  2. [Bars](#)
  3. [SymbolSelect](#)
  4. [SymbolsSynchronized](#)
  5. [SymbolInfoDouble](#)
  6. [SymbolInfoInteger](#)
  7. [SymbolInfoString](#)
  8. [SymbolInfoTick](#)
  9. [SymbolInfoSessionQuote](#)
  10. [SymbolInfoSessionTrade](#)
  11. [MarketBookAdd](#)
  12. [MarketBookGet](#)
- запрос к таймсерии по паре символ/период функциями:
  1. [CopyBuffer](#)
  2. [CopyRates](#)
  3. [CopyTime](#)
  4. [CopyOpen](#)
  5. [CopyHigh](#)
  6. [CopyLow](#)
  7. [CopyClose](#)
  8. [CopyTickVolume](#)

### 9. [CopyRealVolume](#)

### 10. [CopySpread](#)

В тот момент, когда происходит первое обращение к чужому символу, процесс тестирования останавливается и происходит подкачка истории по паре символ/период от терминала к агенту тестирования. Одновременно включается генерация тиковой последовательности для этого символа.

Для каждого инструмента генерируется собственная тиковая последовательность в соответствие с выбранным режимом генерации тиков. Кроме того, можно явно запросить историю для нужных символов с помощью вызова функции [SymbolSelect\(\)](#) в обработчике OnInit() - загрузка истории будет произведена сразу же до начала тестирования советника.

Таким образом, для проведения мультивалютного тестирования в клиентском терминале MetaTrader 5 не требуется предпринимать никаких дополнительных усилий. Достаточно открыть графики соответствующих инструментов в клиентском терминале. История по нужным символам будет автоматически загружена с торгового сервера при условии, что эти данные есть на нем.

## Моделирование времени в тестере

При тестировании локальное время [TimeLocal\(\)](#) всегда равно серверному времени [TimeTradeServer\(\)](#). В свою очередь, серверное время всегда равно времени, соответствующему времени GMT - [TimeGMT\(\)](#). Таким образом, все эти функции при тестировании выдают одно и то же время.

Отсутствие разницы между GMT, локальным и серверным временем в тестере сделано сознательно по той самой причине, что связь с сервером может быть не всегда. А результаты тестирования должны быть одинаковыми, независимо от наличия связи. Информация о серверном времени не хранится локально, а берётся с сервера.

## Графические объекты при тестировании

Во время тестирования/оптимизации не осуществляется построение графических объектов. Таким образом, при обращении к свойствам созданного объекта во время тестирования/оптимизации эксперт получит нулевые значения.

Данное ограничение не распространяется на тестирование в визуальном режиме.

## Функция OnTimer() в тестере

В MQL5 возможна обработка событий таймера. Вызов обработчика [OnTimer\(\)](#) производится независимо от режима тестирования. Это означает, что если тестирование запущено в режиме "Только цены открытия" на периоде H4 и внутри эксперта установлен таймер с вызовом каждую секунду, то на открытии каждого H4 бара один раз будет вызван обработчик OnTick() и 14400 раз (3600 секунд \* 4 часа) в течение бара будет вызван обработчик OnTimer(). Насколько при этом увеличится время тестирования эксперта, зависит от логики эксперта.

Для проверки зависимости времени тестирования от заданной периодичности таймера был написан простой эксперт без торговых операций.

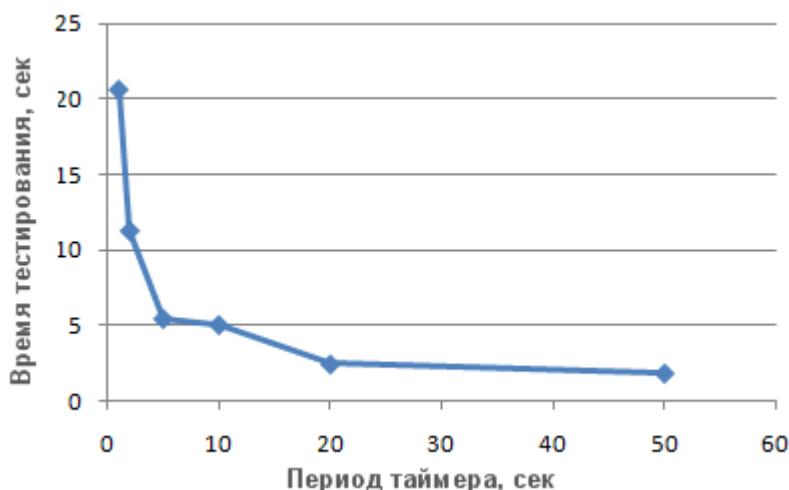
```
//--- input parameters
```

```

input int      timer=1;           // значение таймера, сек
input bool     timer_switch_on=true; // таймер включен
//+-----+
//| Expert initialization function          |
//+-----+
int OnInit()
{
//--- запустим таймер если timer_switch_on==true
    if(timer_switch_on)
    {
        EventSetTimer(timer);
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function         |
//+-----+
void OnDeinit(const int reason)
{
//--- остановим таймер
    EventKillTimer();
}
//+-----+
//| Timer function                           |
//+-----+
void OnTimer()
{
//---
// ничего не делаем, тело обработчика пустое
}
//+-----+

```

Были сделаны замеры времени тестирования при различных значениях параметра timer (периодичность события Timer). На полученных данных построен график зависимости времени тестирования T от значения периодичности Period.



Хорошо видно, чем меньше параметр timer при инициализации таймера функцией [EventSetTimer\(timer\)](#), тем меньше период (Period) между вызовами обработчика OnTimer(), и тем больше время тестирования Т при одних и тех же остальных условиях.

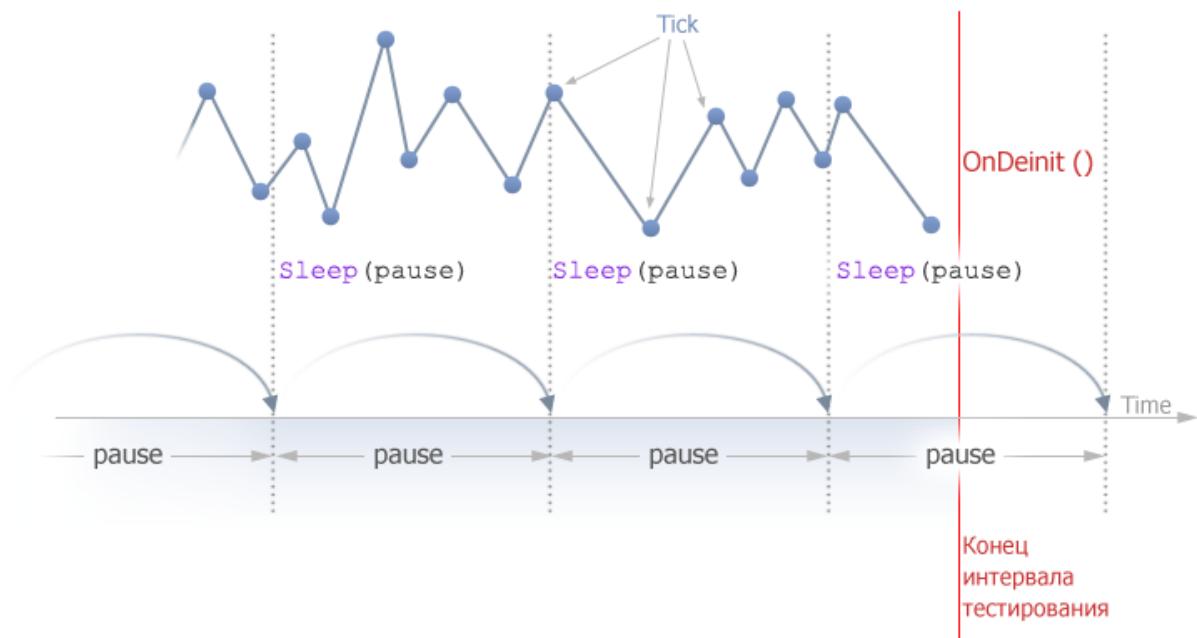
## Функция Sleep() в тестере

Функция [Sleep\(\)](#) позволяет в эксперте или скрипте приостановить выполнение mq5-программы на некоторое время при работе на графике. Это может понадобиться при запросе каких-либо данных, которые в момент запроса еще не готовы и необходимо дождаться момента их готовности. Подробный пример использования функции Sleep() можно посмотреть в разделе [Организация доступа к данным](#).

В тестере же вызовы Sleep() не задерживают процесс тестирования. При вызове Sleep() "проигрываются" сгенерированные тики в пределах указанной задержки, в результате чего могут сработать отложенные ордера, стопы и т.д. После вызова Sleep() смоделированное в тестере время увеличивается на интервал, указанный в параметре функции Sleep.

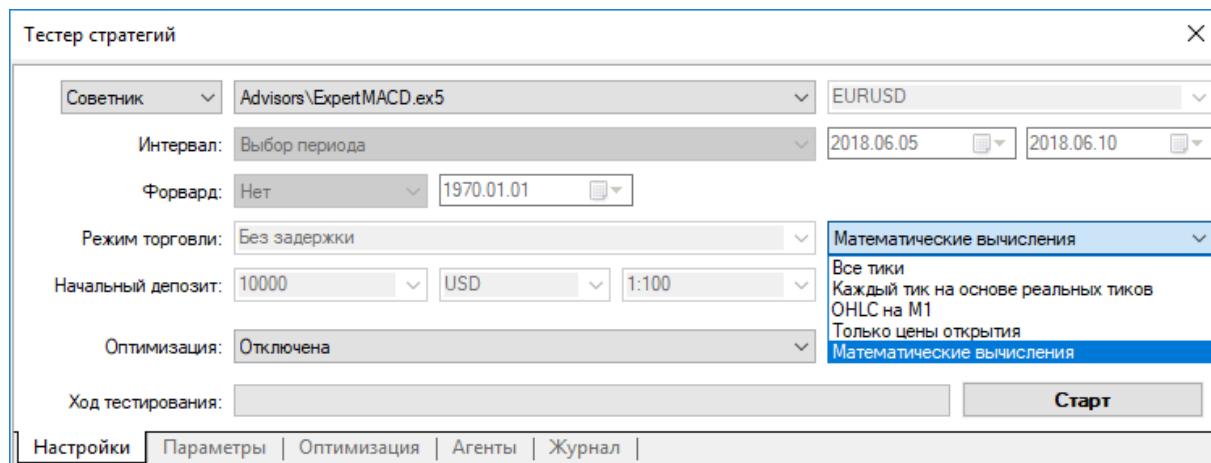
Если в результате выполнения функции Sleep() текущее время в тестере вышло за конец периода тестирования, то будет получена ошибка "бесконечный цикл в Sleep". При получение такой ошибки результаты тестирования не отбрасываются, все вычисления производятся в полном объеме (количество сделок, просадка и т.д.) и результаты данного тестирования передаются терминалу.

Функция Sleep() не будет работать в OnDeinit(), так как после ее вызова тестерное время гарантированно окажется за пределами интервала тестирования.



## Использование тестера для задач оптимизации в математических вычислениях

Тестер в терминале MetaTrader 5 можно использовать не только для проверки торговых стратегий, но и для математических расчётов. Для этого необходимо выбрать соответствующий режим в настройках:



При выборе режима "Математические вычисления" будет произведен "пустой" прогон агента тестирования. Пустой прогон означает, что не будет производиться генерация тиков и загрузки истории. При таком прогоне будут вызваны только три функции: OnInit(), OnTester(), OnDeinit().

Если дата окончания тестирования меньше или равна дате начала тестирования, то это также будет означать тестирования в режиме "Математические вычисления".

При использовании тестера для решения математических задач закачка истории и генерация тиков не происходят.

Типичная математическая задача для решения в тестере MetaTrader 5 - поиск экстремума от функции многих переменных. Для ее решения необходимо:

- Разместить блок вычислений значения функции от многих переменных в `OnTester()` и вернуть вычисленное значение через `return(значение_функции);`
- Вынести параметры функции в глобальную область программы в виде `input-переменных`;

Компилируем советник, открываем окно "Тестер". На вкладке "Входные параметры" отмечаем требуемые входные переменные и задаем для них задаем границы в пространстве значений и шаг для перебора.

Выбираем тип оптимизации - "Медленный (полный перебор параметров)" или "Быстрая (генетический алгоритм)". Для простого поиска экстремума функции лучше выбрать быструю оптимизацию, но если нужно вычислить значения на всем пространстве переменных, то подойдет медленная оптимизация.

Выбираем режим "Математические вычисления" и запускаем кнопкой "Старт" процедуру оптимизации. Необходимо помнить, что при оптимизации всегда ищется локальный максимум значения функции `OnTester`. Для поиска локального минимума можно из функции `OnTester` возвращать значение, обратное вычисленному значению функции:

```
return(1/значение_функции);
```

При этом необходимо самостоятельно реализовать проверку, чтобы значение\_функции не было равно нулю, так как в противном случае можно получить критическую ошибку деления на ноль. Есть и другой вариант, более подходящий и не искажающий результаты оптимизации, он предложен читателями статьи:

```
return(-значение_функции);
```

В этом варианте не требуется проверять значение\_функции на равенство нулю и сама поверхность результатов оптимизации в 3D-представлении имеет ту же форму, только зеркально отраженную от исходной.

В качестве примера приведем функцию `sink()`:

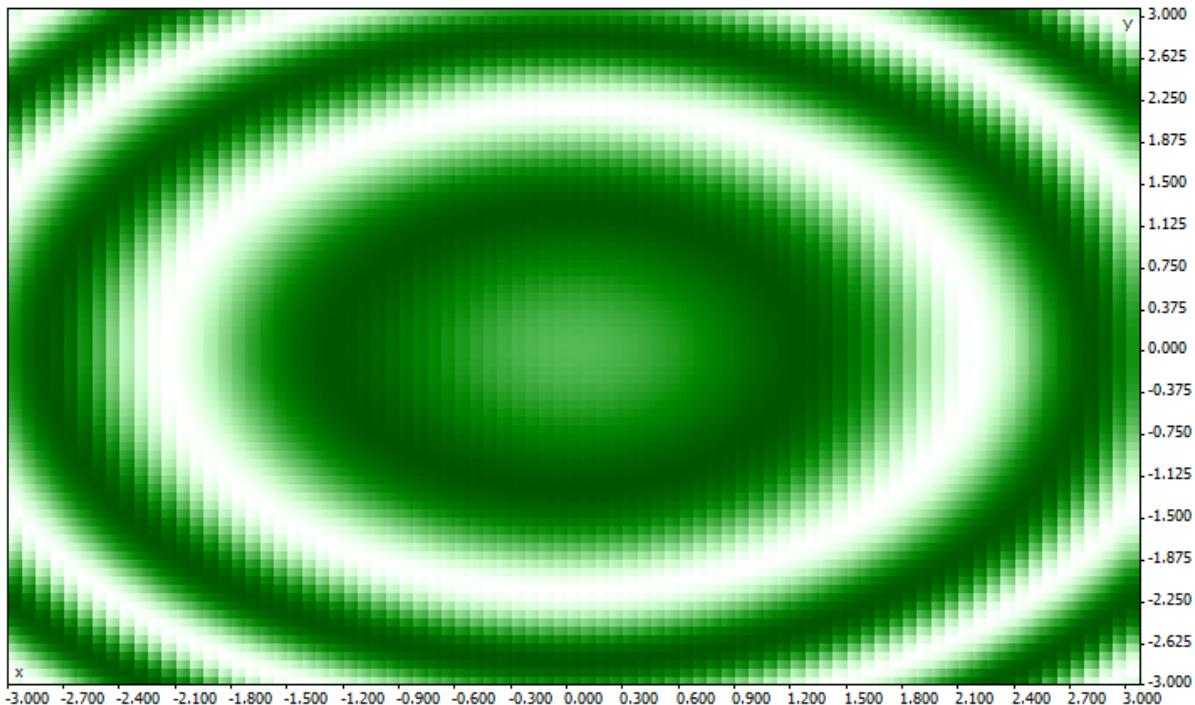
$$\text{sink}(x, y) = \sin(x^2 + y^2)$$

Код советника для поиска экстремума этой функции поместим в `OnTester()`:

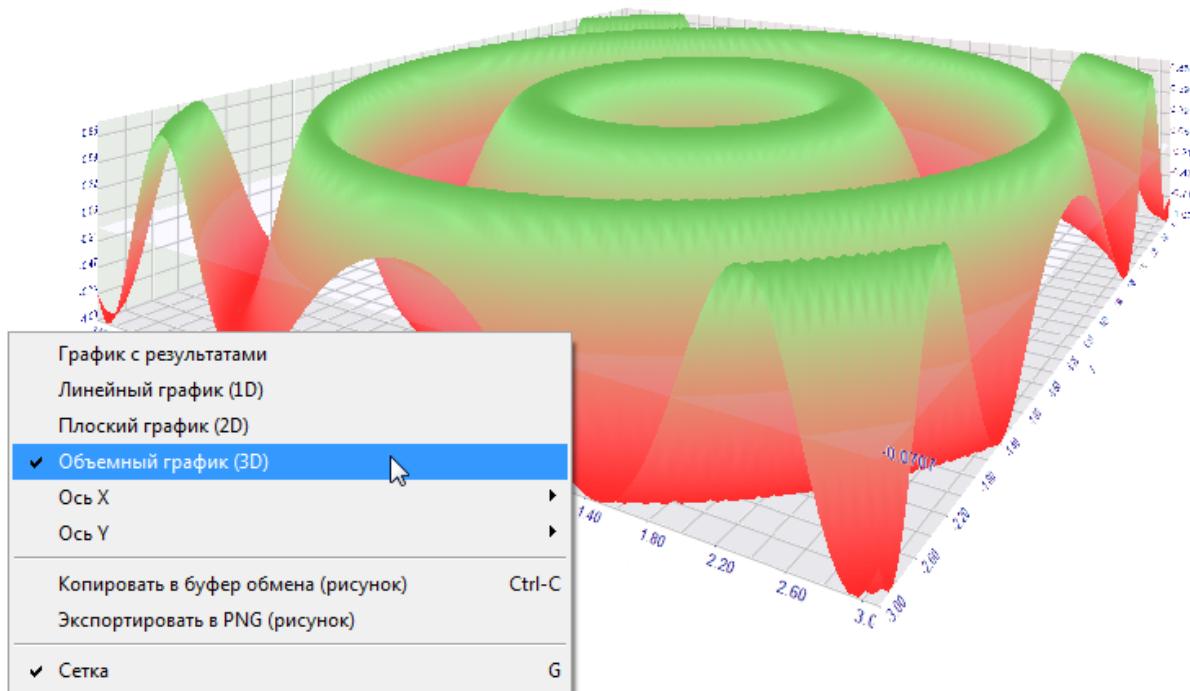
```
//-----+
//|                                                 Sink.mq5 |
//|                                                 Copyright 2011, MetaQuotes Software Corp. |
//|                                                 https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input double     x=-3.0; // start=-3, step=0.05, stop=3
input double     y=-3.0; // start=-3, step=0.05, stop=3
//+-----+
//| Tester function
```

```
//+-----+
double OnTester()
{
//---
    double sink=MathSin (x*x+y*y) ;
//---
    return(sink) ;
}
//-----+
```

Проведем оптимизацию и представим [результаты оптимизации](#) в виде 2D графика.



Чем лучше значение для заданной пары параметров  $(x,y)$ , тем более насыщенный цвет. Как и ожидалось из вида формулы функции  $\text{sink}()$ , ее значения образуют концентрические круги с центром в точке  $(0,0)$ . Для функции  $\text{sink}()$  не существует абсолютного экстремума. Это хорошо видно при просмотре результатов оптимизации в режиме 3D:



## Синхронизация баров при тестировании в режиме "Только цены открытия"

Тестер в клиентском терминале MetaTrader 5 позволяет проверять и, так называемые, "мультивалютные" советники. Мультивалютный советник - это советник, который торгует на двух или более символах.

Тестирование стратегий, торгующих на нескольких инструментах, налагает на тестер несколько дополнительных технических требований:

- генерации тиков для этих инструментов;
- расчет значений индикаторов для этих инструментов;
- расчет маржевых требований по этим инструментам;
- синхронизация сгенерированных тиковых последовательностей по всем торгуемым инструментам.

Тестер генерирует и проигрывает для каждого инструмента тиковую последовательность в соответствии с выбранным режимом торговли. При этом новый бар на каждом инструменте открывается независимо от того, как открылся бар на другом инструменте. Это означает, что при тестировании мультивалютного эксперта возможна ситуация (и чаще всего так и бывает), когда на одном инструменте новый бар уже открылся, а на другом еще нет. Таким образом, при тестировании все происходит как в жизни.

Такое достоверное моделирование развития истории в тестере не вызывает вопросов до тех пор, пока используются режимы тестирования "Все тики" и "1 minute OHLC". При этих режимах в пределах одной свечи генерируется достаточное количество тиков, чтобы дождаться момента синхронизации баров с разных символов. Но как тестировать мультивалютные стратегии в режиме "Только цены открытия", если требуется обязательная синхронизация баров на торгуемых

инструментах? Ведь в этом режиме эксперт вызывается только на одном тике, который соответствует времени открытия бара.

Поясним на примере: если мы тестируем эксперта на символе EURUSD, и на EURUSD открылась новая часовая свеча, то мы легко узнаем этот факт - при тестировании в режиме "Только цены открытия" событие [NewTick](#) соответствует моменту открытия бара на тестируемом периоде. Но при этом нет никакой гарантии, что новая свеча открылась по символу GBPUSD, который используется в эксперте.

В обычных условиях достаточно завершить работу функции [OnTick\(\)](#) и проверить появление нового бара на GBPUSD на следующем тике. Но при тестировании в режиме "Только цены открытия" другого тика не будет, и может показаться, что этот режим не годится для тестирования мультивалютных экспертов. Но это не так - не забывайте, что тестер в MetaTrader 5 ведет себя также, как и в жизни. Можно дождаться момента, когда на другом символе откроется новый бар с помощью функции [Sleep\(\)](#)!

Код советника `Synchronize_Bars_Use_Sleep.mq5`, который демонстрирует пример синхронизации баров при тестировании в режиме "Только цены открытия":

```

//+-----+
//|                               Synchronize_Bars_Use_Sleep.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input string    other_symbol="USDJPY";
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- сверим текущий символ
if(_Symbol==other_symbol)
{
PrintFormat("Необходимо указать другой символ или запустить тестирование на другом");
//--- принудительно прекращаем тестирование эксперта
return(INIT_PARAMETERS_INCORRECT);
}
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Expert tick function
//+-----+
void OnTick()
{
//--- статическая переменная для хранения времени открытия последнего бара

```

```

        static datetime last_bar_time=0;
//--- признак того, что время открытия последнего бара на разных символах синхронизировано
        static bool synchronized=false;
//--- если статическая переменная еще неинициализирована
if(last_bar_time==0)
{
//--- это первый вызов, запишем время открытия и выйдем
last_bar_time=(datetime)SeriesInfoInteger(_Symbol,Period(),SERIES_LASTBAR_DATE);
PrintFormat("Инициализировали переменную last_bar_time значением %s",TimeToString(last_bar_time));
}
//--- получим время открытия последнего бара по своему символу
datetime curr_time=(datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_LASTBAR_DATE);
//--- если время открытия текущего бара не совпадает с тем, что хранится в last_bar_time
if(curr_time!=last_bar_time)
{
//--- запомним время открытия нового бара в статической переменной
last_bar_time=curr_time;
//--- синхронизация нарушена, сбросим флаг в false
synchronized=false;
//--- выведем сообщение об этом событии
PrintFormat("На символе %s открылся новый бар в %s",_Symbol,TimeToString(TimeCurrent()));
}
//--- сюда будем сохранять время открытия бара на чужом символе
datetime other_time;
//--- цикл, пока время открытия последнего бара по другому символу не сравняется с текущим
while(!(curr_time==(other_time=(datetime)SeriesInfoInteger(other_symbol,Period(),SERIES_LASTBAR_DATE))));
{
PrintFormat("Подождем 5 секунд..");
//--- подождем 5 секунд и опять запросим SeriesInfoInteger(other_symbol,Period())
Sleep(5000);
}
//--- время открытия бара теперь одинаково для обоих символов
synchronized=true;
PrintFormat("Время открытия последнего бара на своем символе %s: %s",_Symbol,TimeToString(last_bar_time));
PrintFormat("Время открытия последнего бара на символе %s: %s",other_symbol,TimeToString(other_time));
//--- TimeCurrent() не подойдет, используем TimeTradeServer() для точности
Print("Бары синхронизировались в ",TimeToString(TimeTradeServer()),TIME_DATE|TIME_SECONDS);
}
//+-----+

```

Обратите внимание на последнюю строку в советнике, которая выводит текущее время, когда был установлен факт синхронизации:

```
Print("Бары синхронизировались в ",TimeToString(TimeTradeServer()),TIME_SECONDS));
```

Для вывода текущего времени мы использовали функцию [TimeTradeServer\(\)](#), а не [TimeCurrent\(\)](#). Дело в том, что функция [TimeCurrent\(\)](#) возвращает время последнего тика, которое никак не изменилось после использования [Sleep\(\)](#). Запустите советник в режиме "Только цены открытия" и увидите сообщения о синхронизации баров.

Core1	2010.12.01 20:00:05	Бары синхронизировались в 2010.12.01 20:00:05
Core1	2010.12.01 20:00:05	Время открытия последнего бара на символе USDJPY: 2010.12.01 20:00
Core1	2010.12.01 20:00:05	Время открытия последнего бара на своем символе EURUSD: 2010.12.01 20:00
Core1	2010.12.01 20:00:00	Подождем 5 секунд..
Core1	2010.12.01 20:00:00	На символе EURUSD открылся новый бар в 2010.12.01 20:00
Core1	2010.12.01 16:00:05	Бары синхронизировались в 2010.12.01 16:00:05

Используйте функцию `TimeTradeServer()` вместо `TimeCurrent()`, если требуется получить текущее серверное время, а не время поступления последнего тика.

Есть и другой способ синхронизации баров - с помощью таймера. Пример такого эксперта `Synchronize_Bars_Use_OnTimer.mq5` приложен к статье.

## Функция `IndicatorRelease()` в тестере

После окончания одиночного тестирования автоматически открывается график инструмента, на котором отображаются совершенные сделки и индикаторы, которые использовались в эксперте. Это помогает визуально проверить моменты входа и выхода, а также сопоставить их со значениями индикаторов.

**Важно:** индикаторы, отображаемые на автоматически открытом после завершения тестирования графике, рассчитываются заново уже после окончания тестирования. Даже если эти индикаторы использовались в тестируемом эксперте.

Но в некоторых случаях программисту может понадобиться скрыть информацию о том, какие индикаторы задействованы в торговом алгоритме. Например, код эксперта сдается в аренду или продается в виде исполняемого файла без предоставления исходного кода. Для этих целей подойдет функция `IndicatorRelease()`.

Если в терминале задан шаблон с названием `tester.tpl` в каталоге `/profiles/templates` клиентского терминала, то именно он будет применен к открываемому графику. При его отсутствии применяется шаблон по умолчанию (`default.tpl`).

Функция `IndicatorRelease()` изначально предназначена для освобождения расчетной части индикатора, если он больше не нужен. Это позволяет экономить как память, так и ресурсы процессора, потому что каждый тик вызывает расчет индикатора. Второе ее предназначение - запретить показ индикатора на графике тестирования после окончания одиночного прогона.

Чтобы запретить показ индикатора на графике по окончании тестирования, вызовете `IndicatorRelease()` с хэндлом индикатора в обработчике `OnDeinit()`. Функция `OnDeinit()` всегда вызывается после завершения и перед показом графика тестирования.

```
//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//---
    bool hidden=IndicatorRelease(handle_ind);
    if(hidden)
        Print("IndicatorRelease() выполнена успешно");
}
```

```

    else
        Print("IndicatorRelease() вернула false. Код ошибки ", GetLastError());
}

```

Для того чтобы запретить показ индикатора на графике после завершения одиночного тестирования, используйте функцию `IndicatorRelease()` в обработчике `OnDeinit()`.

## Обработка событий в тестере

Наличие обработчика `OnTick()` в эксперте не является обязательным для того чтобы его можно было подвергнуть проверке на исторических данных в тестере терминала MetaTrader 5. Достаточно того, чтобы в советнике была хотя бы одна функция-обработчик из перечисленных:

- [OnTick\(\)](#) - обработчик события прихода нового тика;
- [OnTrade\(\)](#) - обработчик торгового события;
- [OnTimer\(\)](#) - обработчик события прихода сигнала от таймера;
- [OnChartEvent\(\)](#) - обработчик пользовательских событий.

При тестировании в эксперте можно обрабатывать пользовательские события с помощью функции [OnChartEvent\(\)](#), но в индикаторах эта функция в тестере не вызывается. Даже если индикатор имеет обработчик [OnChartEvent\(\)](#) и этот индикатор используется в тестируемом эксперте, то сам индикатор не будет получать никаких пользовательских событий.

Индикатор при тестировании может генерировать пользовательские события с помощью функции [EventChartCustom\(\)](#), а советник может обрабатывать это событие в `OnChartEvent()`.

Помимо вышеуказанных событий в тестере стратегий генерируются специальные события, связанные с процессом тестирования и оптимизации:

- Tester - данное событие генерируется по окончании тестирования эксперта на исторических данных. Обработка события `Tester` производится функцией [OnTester\(\)](#). Эта функция может быть использована только в экспертах при тестировании и предназначена в первую очередь для расчета некоторого значения, используемого в качестве критерия `Custom max` при генетической оптимизации входных параметров.
- `TesterInit` - данное событие генерируется при запуске оптимизации в тестере стратегий перед самым первым проходом. Обработка события `TesterInit` производится функцией [OnTesterInit\(\)](#). Эксперт, имеющий данный обработчик, при запуске оптимизации автоматически загружается на отдельном графике терминала с указанными в тестере символом и периодом, и получает событие `TesterInit`. Функция предназначена для инициализации эксперта перед началом оптимизации для последующей [обработки результатов оптимизации](#).
- `TesterPass` - данное событие генерируется при поступлении нового [фрейма данных](#). Обработка события `TesterPass` производится функцией [OnTesterPass\(\)](#). Эксперт с данным обработчиком автоматически загружается на отдельном графике терминала с указанными для тестирования символом/периодом и получает во время оптимизации события `TesterPass` при получении фрейма. Функция предназначена для динамической обработки [результатов оптимизации](#) прямо "на лету", не дожидаясь её окончания. Добавление фреймов производится функцией [FrameAdd\(\)](#), которую можно вызывать по окончании одиночного прохода в обработчике [OnTester\(\)](#).
- `TesterDeinit` - данное событие генерируется по окончании оптимизации эксперта в тестере стратегий. Обработка события `TesterDeinit` производится функцией [OnTesterDeinit\(\)](#). Эксперт с данным обработчиком автоматически загружается на график при запуске оптимизации и

получает событие TesterDeinit после её завершения. Функция предназначена для финальной обработки всех [результатов оптимизации](#).

## Агенты тестирования

Тестирование в клиентском терминале MetaTrader 5 осуществляется с помощью [агентов тестирования](#). Локальные агенты создаются и подключаются автоматически. Количество локальных агентов по умолчанию соответствует количеству ядер на компьютере.

У каждого агента тестирования своя копия [глобальных переменных](#), которая никак не связана с клиентским терминалом. Сам терминал является диспетчером, который раздает задачи локальным и удаленным агентам. После выполнения очередного задания по тестированию советника с заданными параметрами агент возвращает терминалу результаты. При одиночном тестировании используется только один агент.

Агент хранит полученную от терминала историю в отдельных папках по имени инструмента, то есть история для EURUSD хранится в папке с именем EURUSD. Кроме того история инструментов разделяется по источникам. Структура для хранения истории выглядит таким образом:

```
каталог_тестера\Agent-IPaddress-Port\bases\имя_источника\history\имя_инструмента
```

Например, история по EURUSD с сервера MetaQuotes-Demo может храниться в папке каталог\_тестера\Agent-127.0.0.1-3000\bases\MetaQuotes-Demo\EURUSD.

Локальный агент после окончания тестирования находится в режиме ожидания следующей задачи в течение 5 минут, чтобы не терять время на запуск при следующих вызовах. Только по истечении ожидания локальный агент прекращает свою работу и выгружается из памяти компьютера.

При досрочном завершении тестирования со стороны пользователя (кнопка "Отмена"), а также при закрытии клиентского терминала все локальные агенты тут же прекращают свою работу и выгружаются из памяти.

## Обмен данными между терминалом и агентом

При запуске тестирования терминал готовит для отправки агенту несколько блоков параметров:

- Входные параметры тестирования (режим моделирования, интервал тестирования, инструмент, критерий оптимизации и т.д.)
- Список выбранных в "Обзоре рынка" инструментов
- Спецификация тестируемого инструмента (размер контракта, допустимые отступы от рынка для установки StopLoss и TakeProfit, и т.д.)
- Тестируемый эксперт и значения его входных параметров
- Информация о дополнительных файлах (библиотеки, индикаторы, файлы данных - [#property tester\\_...](#))

tester_indicator	<a href="#">string</a>	Имя пользовательского индикатора в формате "имя_индикатора.ex5". Необходимые для тестирования индикаторы определяются автоматически
------------------	------------------------	--

		из вызова функций <a href="#">iCustom()</a> , если соответствующий параметр задан константной строкой. Для остальных случаев (использование функции <a href="#">IndicatorCreate()</a> или использование неконстантной строки в параметре, задающем имя индикатора) необходимо данное свойство
tester_file	<a href="#">string</a>	Имя файла для тестера с указанием расширения, заключенное в двойные кавычки (как константная строка). Указанный файл будет передан тестеру в работу. Входные файлы для тестирования, если необходимы, должны указываться всегда
tester_library	<a href="#">string</a>	Имя библиотеки с расширением, заключенное в двойные кавычки. Библиотека может быть как с расширением dll, так и с расширением ex5. Необходимые для тестирования библиотеки определяются автоматически. Однако, если какая-либо библиотека используется <a href="#">пользовательским</a> индикатором, то необходимо использовать данное свойство

Для каждого блока параметров создается цифровой отпечаток в виде MD5-хэша, который и посылается агенту. MD5-хэш является уникальным для каждого набора, его объем во много раз меньше объема информации, на основе которой он вычислен.

Агент получает хэши блоков и сравнивает с теми, что он уже хранит у себя. Если отпечаток данного блока параметров отсутствует у агента, или присланный хэш отличается от имеющегося, то агент запрашивает сам блок параметров. Таким образом уменьшается трафик между терминалом и агентом.

После проведения тестирования агент возвращает терминалу все результаты прогона, показываемые во вкладках "Результаты тестирования" и "Результаты оптимизации": полученная прибыль, количество сделок, коэффициент Шарпа, результат функции OnTester() и т.д.

При оптимизации терминал раздает агентам задачи на проведение тестирования небольшими пакетами, в каждом пакете находится несколько заданий (каждое задание означает одиночное тестирование с набором входных параметров). Это уменьшает время обмена между терминалом и агентом.

Агенты никогда не записывают на жесткий диск полученные от терминала EX5-файлы (эксперт, индикаторы, библиотеки и т.д.) из соображений безопасности, чтобы на компьютере с установленным агентом нельзя было воспользоваться присланными данным. Все остальные файлы, в том числе DLL, записываются в "песочницу". В удаленных агентах нельзя тестировать экспертов с использованием DLL.

Результаты тестирования складываются терминалом в специальный кэш результатов (результатирующий кэш) для последующего быстрого доступа к ним при необходимости. Для каждого набора параметров терминал ищет в результатеющем кэше уже готовые результаты от предыдущих запусков для исключения повторных запусков. Если результат с таким набором параметров не найден, агенту отдается задание на проведение тестирования.

Весь трафик между терминалом и агентами шифруется.

Тики не пересыпаются по сети, они генерируются на тестерных агентах.

## Использование общей папки всех клиентских терминалов

Все тестерные агенты изолированы друг от друга и от клиентского терминала: у каждого агента есть собственная папка, в которую записываются логи агента. Кроме того, все файловые операции при тестировании агента происходят в папке `имя_агента/MQL5/Files`. Однако можно реализовать взаимодействие между локальными агентами и клиентским терминалом через общую папку всех клиентских терминалов, если при открытии файла указать флаг `FILE_COMMON`:

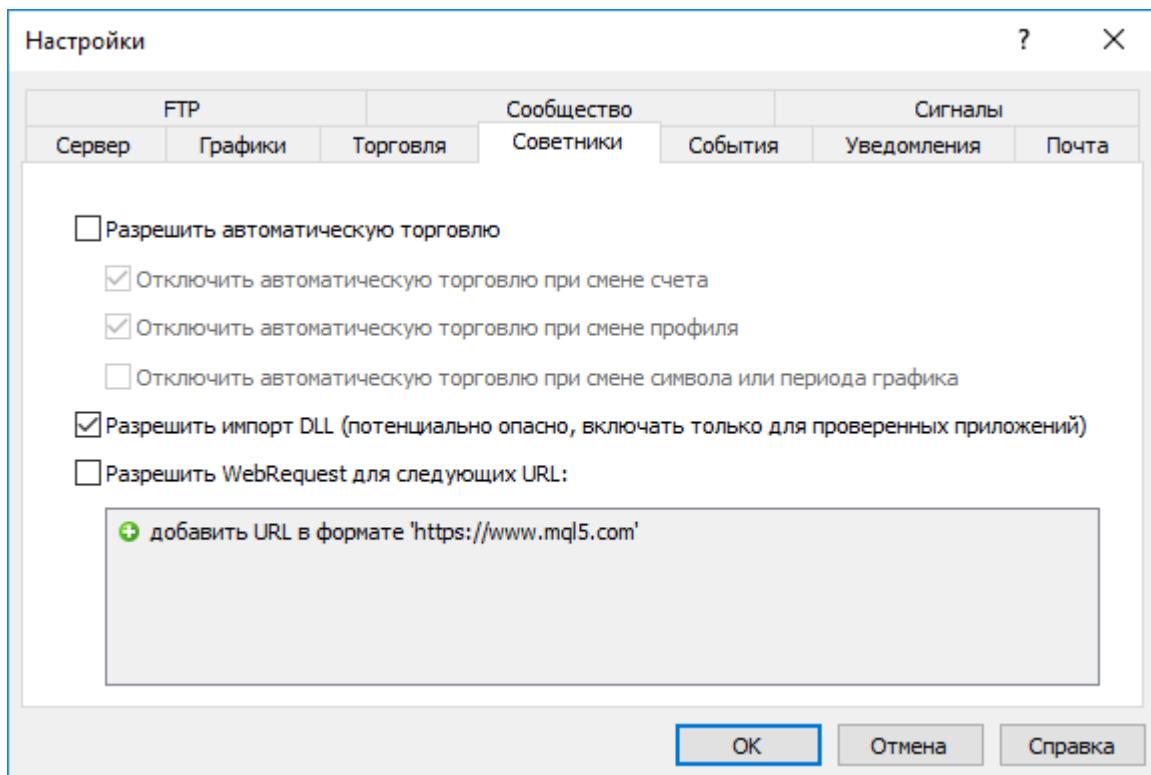
```
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
    //--- общая папка всех клиентских терминалов
    common_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
    //--- выведем имя этой папки
    PrintFormat("Откроем файл в общей папке клиентских терминалов %s", common_folder);
    //--- откроем файл в общей папке (указан флаг FILE_COMMON)
    handle=FileOpen(filename,FILE_WRITE|FILE_READ|FILE_COMMON);
    ... дальнейшие действия
    //---
    return(INIT_SUCCEEDED);
}
```

## Использование DLL

Для ускорения оптимизации можно использовать не только локальные, но и [удаленные агенты](#). При этом есть некоторые ограничения для удаленных агентов. Во-первых, удаленные агенты не выводят в свои логи результаты выполнения функции `Print()`, сообщения об открытии/закрытии

позиций. Выводится в лог минимум информации чтобы неправильно написанные эксперты не забили сообщениями жесткий диск компьютера, на котором работает удаленный агент.

Второе ограничение - запрет на использование DLL при тестировании экспертов. Вызовы DLL безусловно запрещены на удалённых агентах из соображений безопасности. На локальных агентах вызовы dll в тестируемых экспертах разрешены только в том случае, если установлено соответствующее разрешение "Разрешить импорт DLL".



**Важно:** при использовании полученных со стороны советников (скриптов, индикаторов), которые требуют разрешить вызовы DLL, вы должны осознавать весь риск, который принимаете на себя в случае разрешения этой опции в настройках терминала. Независимо от того, как будет использован советник - для тестирования или для запуска на графике.

## Предопределенные переменные

Для каждой выполняющейся MQL5-программы поддерживается ряд предопределенных переменных, которые отражают состояние текущего ценового графика на момент запуска программы - эксперта, скрипта или пользовательского индикатора.

Значение предопределенным переменным устанавливает клиентский терминал перед запуском MQL5-программы на выполнение. Предопределенные переменные константы и не могут быть изменены из MQL5-программы. Исключение составляет переменная `_LastError`, которая может быть обнулена функцией [ResetLastError](#).

Переменная	Значение
<a href="#"><code>_AppliedTo</code></a>	Переменная <code>_AppliedTo</code> позволяет узнать в индикаторе тип данных, на которых он рассчитывается
<a href="#"><code>_Digits</code></a>	Количество десятичных знаков после запятой
<a href="#"><code>_Point</code></a>	Размер пункта текущего инструмента в валюте котировки
<a href="#"><code>_LastError</code></a>	Значение последней ошибки
<a href="#"><code>_Period</code></a>	Значение таймфрейма текущего графика
<a href="#"><code>_RandomSeed</code></a>	Текущее состояние генератора псевдослучайных целых чисел
<a href="#"><code>_StopFlag</code></a>	Флаг остановки программы
<a href="#"><code>_Symbol</code></a>	Имя символа текущего графика
<a href="#"><code>_UninitReason</code></a>	Код причины deinициализации программы
<a href="#"><code>_IsX64</code></a>	Переменная <code>_IsX64</code> позволяет узнать в каком терминале запущена MQL5-программа

Предопределенные переменные не могут быть определены в библиотеке. Библиотека использует эти переменные, определенные в программе, из которой эта библиотека была вызвана.

## int \_AppliedTo

Переменная `_AppliedTo` позволяет узнать в индикаторе тип данных, на которых он рассчитывается:

Тип данных	Значение	Описание данных, используемых для расчета индикатора
—	0	Индикатор использует вторую форму вызова <a href="#">OnCalculate()</a> - данные для расчета не задаются одним каким-то буфером или массивом данных
Close	1	Цены Close
Open	2	Цены Open
High	3	Цены High
Low	4	Цены Low
Median Price (HL/2)	5	Средняя цена = (High+Low)/2
Typical Price (HLC/3)	6	Типическая цена = (High+Low+Close)/3
Weighted Price (HLCC/4)	7	Взвешенная цена = (Open+High+Low+Close)/4
Previous Indicator's Data	8	Данные индикатора, который был запущен на график перед данным индикатором
First Indicator's Data	9	Данные индикатора, который был запущен на график самым первым
Indicator handle	10+	Данные с индикатора, переданного в функцию <a href="#">iCustom()</a> с помощью хендла индикатора. Значение <code>_AppliedTo</code> содержит хендл индикатора

### Пример:

```
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
    //--- indicator buffers mapping
```

```

    SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
// получим тип данных, на котором считается индикатор
Print("_AppliedTo=",_AppliedTo);
Print(getIndicatorDataDescription(_AppliedTo));
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Описание данных, на которых расчитывается индикатор |
//+-----+
string getIndicatorDataDescription(int data_id)
{
    string descr="";
    switch(data_id)
    {
        case(0):descr="It's first type of OnCalculate() - no data buffer";
        break;
        case(1):descr="Indicator calculates on Close price";
        break;
        case(2):descr="Indicator calculates on Open price";
        break;
        case(3):descr="Indicator calculates on High price";
        break;
        case(4):descr="Indicator calculates on Low price";
        break;
        case(5):descr="Indicator calculates on Median Price (HL/2)";
        break;
        case(6):descr="Indicator calculates on Typical Price (HLC/3)";
        break;
        case(7):descr="Indicator calculates on Weighted Price (HLCC/4)";
        break;
        case(8):descr="Indicator calculates Previous Indicator's data";
        break;
        case(9):descr="Indicator calculates on First Indicator's data";
        break;
        default: descr="Indicator calculates on data of indicator with handle="+string(handle);
        break;
    }
//---
return descr;
}

```

**Смотри также**

[ENUM\\_APPLIED\\_PRICE](#)

## int \_Digits

В переменной `_Digits` хранится количество десятичных знаков после запятой, определяющее точность измерения цены символа текущего графика.

Можно также использовать функцию [Digits\(\)](#).

## double \_Point

В переменной \_Point хранится размер пункта текущего инструмента в валюте котировки.

Можно также использовать функцию [Point\(\)](#).

## int \_LastError

В переменной `_LastError` хранится значение последней [ошибки](#), произошедшей во время исполнения mq5-программы. Сбросить значение в ноль можно функцией [ResetLastError\(\)](#).

Для получения кода ошибки можно также использовать функцию [GetLastError\(\)](#).

## int \_Period

В переменной `_Period` хранится значение таймфрейма текущего графика.

Можно также использовать функцию [Period\(\)](#).

**Смотри также**

[PeriodSeconds](#), [Периоды графиков](#), [Дата и время](#), [Видимость объектов](#)

## \_RandomSeed

Переменная для хранения текущего состояния при генерации псевдослучайных целых чисел. [\\_RandomSeed](#) меняет своё значение при вызове [MathRand\(\)](#). Для установки нужного начального состояния используйте [MathStrand\(\)](#).

Случайное число `x`, получаемое функцией `MathRand()`, вычисляется при каждом вызове следующим образом:

```
x=_RandomSeed*214013+2531011;  
_RandomSeed=x;  
x=(x>>16)&0x7FFF;
```

Смотри также

[MathRand\(\)](#), [MathStrand\(\)](#), [Целые типы](#)

## bool \_StopFlag

В переменной `_StopFlag` хранится флаг остановки mql5-программы. Когда клиентский терминал пытается остановить программу, в эту переменную записывается значение `true`.

Для проверки значения флага `_StopFlag` можно также использовать функцию [`IsStopped\(\)`](#).

## string \_Symbol

В переменной `_Symbol` хранится имя символа текущего графика.

Можно также использовать функцию [Symbol\(\)](#).

## int \_UninitReason

В переменной \_UninitReason хранится код [причины deinициализации](#) программы.

Обычно код причины deinициализации получают с помощью функции [UninitializeReason\(\)](#).

## int \_IsX64

Переменная `_IsX64` позволяет узнать в каком терминале запущена MQL5-программа: для 32-х битного терминала `_IsX64=0` и для 64-х битного терминала `_IsX64!=0`.

Можно также использовать функцию [TerminalInfoInteger\(TERMINAL\\_X64\)](#).

**Пример:**

```
// проверим в каком терминале запущена программа
Print("_IsX64=", _IsX64);
if(_IsX64)
    Print("Программа ", __FILE__, " запущена в 64-битном терминале");
else
    Print("Программа ", __FILE__, " запущена в 32-битном терминале");
Print("TerminalInfoInteger(TERMINAL_X64)=", TerminalInfoInteger(TERMINAL_X64));
```

**Смотри также**

[MQLInfoInteger](#), [Импорт функций \(#import\)](#)

## Общие функции

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

Функция	Действие
<a href="#">Alert</a>	Выводит сообщение в отдельном окне
<a href="#">CheckPointer</a>	Возвращает тип указателя объекта
<a href="#">Comment</a>	Выводит сообщение в левый верхний угол ценового графика
<a href="#">CryptEncode</a>	Преобразует данные массива-источника в массив-приемник указанным методом
<a href="#">CryptDecode</a>	Производит обратное преобразование данных массива
<a href="#">DebugBreak</a>	Программная точка останова при отладке
<a href="#">ExpertRemove</a>	Прекращает работу эксперта и выгружает его с графика
<a href="#">GetPointer</a>	Возвращает <a href="#">указатель</a> объекта
<a href="#">GetTickCount</a>	Возвращает количество миллисекунд, прошедших с момента старта системы
<a href="#">GetMicrosecondCount</a>	Возвращает количество микросекунд, прошедших с момента начала работы MQL5-программы
<a href="#">MessageBox</a>	Создает и отображает окно сообщений, а также управляет им
<a href="#">PeriodSeconds</a>	Возвращает количество секунд в периоде
<a href="#">PlaySound</a>	Воспроизводит звуковой файл
<a href="#">Print</a>	Выводит сообщение в журнал
<a href="#">PrintFormat</a>	Форматирует и печатает наборы символов и значений в лог-файл в соответствие с заданным форматом
<a href="#">ResetLastError</a>	Устанавливает значение предопределенной переменной <a href="#">LastError</a> в ноль
<a href="#">ResourceCreate</a>	Создает ресурс изображения на основе набора данных
<a href="#">ResourceFree</a>	Удаляет <a href="#">динамически созданный ресурс</a> (освобождает занятую ресурсом память)
<a href="#">ResourceReadImage</a>	Читает данные графического ресурса, <a href="#">созданного функцией ResourceCreate()</a> или <a href="#">сохраненного в EX5-файле при компиляции</a>

<a href="#">ResourceSave</a>	Сохраняет ресурс в указанный файл
<a href="#">SetUserError</a>	Устанавливает предопределенную переменную _LastError в значение, равное ERR_USER_ERROR_FIRST + user_error
<a href="#">SetReturnError</a>	Устанавливает код возврата, который вернёт процесс терминала при завершение работы
<a href="#">Sleep</a>	Задерживает выполнение текущего эксперта или скрипта на определенный интервал
<a href="#">TerminalClose</a>	Посыпает терминалу команду на завершение работы
<a href="#">TesterHideIndicators</a>	Задает режим показа/скрытия индикаторов, которые используются в эксперте
<a href="#">TesterStatistics</a>	Возвращает значение указанного статистического показателя, рассчитанного по результатам тестирования
<a href="#">TesterStop</a>	Отдает команду на завершении работы программы при <a href="#">тестировании</a> .
<a href="#">TesterDeposit</a>	Эмуляции операций внесения средств в процессе тестирования. Может быть использована в некоторых системах управления капиталом.
<a href="#">TesterWithdrawal</a>	Эмуляция операций снятия средств в процессе тестирования
<a href="#">TranslateKey</a>	Возвращает Unicode-символ по виртуальному коду клавиши
<a href="#">ZeroMemory</a>	Обнуляет переменную, переданную по ссылке. Тип переменной может быть любым, исключение составляют только классы и структуры, имеющие конструкторы

## Alert

Отображает диалоговое окно, содержащее пользовательские данные.

```
void Alert(  
    argument,           // первое значение  
    ...                // последующие значения  
);
```

### Параметры

*argument*

[in] Любые значения, разделенные запятыми. Для разделения выводимой информации на несколько строк можно использовать символ перевода строки "\n" либо "\r\n". Количество параметров не может превышать 64.

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

Массивы нельзя передавать в функцию Alert(). Массивы должны выводиться поэлементно. Данные типа double выводятся с 8 десятичными цифрами после точки, данные типа float выводятся с 5 десятичными цифрами после точки. Для вывода вещественных чисел с другой точностью либо в научном формате необходимо использовать функцию [DoubleToString\(\)](#).

Данные типа bool выводятся в виде строк "true" или "false". Даты выводятся в виде YYYY.MM.DD HH:MI:SS. Для вывода даты в другом формате необходимо использовать функцию [TimeToString\(\)](#). Данные типа color выводятся либо в виде строки R,G,B, либо в виде названия цвета, если этот цвет присутствует в наборе цветов.

При работе в [тестере стратегий](#) функция Alert() не выполняется.

## CheckPointer

Возвращает тип [указателя](#) объекта.

```
ENUM_POINTER_TYPE CheckPointer(
    object* anyobject      // указатель объекта
);
```

### Параметры

*anyobject*  
[in] Указатель объекта.

### Возвращаемое значение

Возвращает значение из перечисления [ENUM\\_POINTER\\_TYPE](#).

### Примечание

Попытка обращения к некорректному указателю приводит к [критическому завершению](#) программы. Поэтому существует необходимость использования функции CheckPointer перед использованием указателя. Указатель может быть некорректным в следующих случаях:

- указатель равен [NULL](#);
- если объект был уничтожен при помощи оператора [delete](#).

Данную функцию можно использовать как проверку указателя на корректность. Значение, отличное от нуля, гарантирует, что по этому указателю можно получить доступ к данным.

### Пример:

```
//+-----+
//| Уничтожает список через уничтожение элементов |
//+-----+
void CMyList::Destroy()
{
//--- служебный указатель для работы в цикле
CItem* item;
//--- пройдемся в цикле и попытаемся удалить динамические указатели
while(CheckPointer(m_items)!=POINTER_INVALID)
{
    item=m_items;
    m_items=m_items.Next();
    if(CheckPointer(item)==POINTER_DYNAMIC)
    {
        Print("Dynamyc object ",item.Identifier()," to be deleted");
        delete (item);
    }
    else
        Print("Non-dynamic object ",item.Identifier()," cannot be deleted");
}
//---
```

{}

**Смотри также**[Указатели объектов](#), [Проверка указателя объекта](#), [Оператор уничтожения объекта delete](#)

## Comment

Выводит комментарий, определенный пользователем, в левый верхний угол графика.

```
void Comment(
    argument,      // первое значение
    ...           // последующие значения
);
```

### Параметры

...

[in] Любые значения, разделенные запятыми. Для разделения выводимой информации на несколько строк можно использовать символ перевода строки "\n" либо "\r\n". Количество параметров не может превышать 64. Общая длина выводимого сообщения (включая служебные неотображаемые символы) не может превышать 2045 символов (лишние символы будут обрезаны при выводе).

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

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

Данные типа double выводятся с точностью до 16 десятичных цифр после точки, при этом данные могут выводиться либо в традиционном либо в научном формате - в зависимости от того, как запись будет наиболее компактна. Данные типа float выводятся с 5 десятичными цифрами после точки. Для вывода вещественных чисел с другой точностью либо в явно указанном формате необходимо использовать функцию [DoubleToString\(\)](#).

Данные типа bool выводятся в виде строк "true" или "false". Даты выводятся в виде YYYY.MM.DD HH:MI:SS. Для вывода даты в другом формате необходимо использовать функцию [TimeToString\(\)](#). Данные типа color выводятся либо в виде строки R,G,B, либо в виде названия цвета, если этот цвет присутствует в наборе цветов.

При работе в [тестере стратегий](#) в режиме оптимизации функция Comment() не выполняется.

### Пример:

```
void OnTick()
{
//---
    double Ask,Bid;
    int Spread;
    Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
    Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
//--- Выведем значения в три строчки
    Comment(StringFormat("Выvodim цены\nAsk = %G\nBid = %G\nSpread = %d",Ask,Bid,Spread));
}
```

### Смотри также

[ChartSetString](#), [ChartGetString](#)

## CryptEncode

Преобразует данные массива-источника в массив-приемник указанным методом.

```
int CryptEncode(
    ENUM_CRYPT_METHOD method,           // метод преобразования
    const uchar&      data[],          // массив-источник
    const uchar&      key[],           // ключ шифрования
    uchar&            result[]);       // массив-приемник
);
```

### Параметры

*method*

[in] Метод преобразования. Может быть одним из значений перечисления [ENUM\\_CRYPT\\_METHOD](#).

*data[]*

[in] Массив-источник.

*key[]*

[in] Ключ шифрования.

*result[]*

[out] Массив-приемник.

### Возвращаемое значение

Количество байт в массиве-приемнике или 0 в случае ошибки. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Пример:

```
//+-----+
//| ArrayToHex
//+-----+
string ArrayToHex(uchar &arr[], int count=-1)
{
    string res="";
//--- проверка размера
    if(count<0 || count>ArraySize(arr))
        count=ArraySize(arr);
//--- преобразование в шестнадцатиричную строку
    for(int i=0; i<count; i++)
        res+=StringFormat("%.2X", arr[i]);
//---
    return(res);
}
//+-----+
//| Script program start function
//+-----+
void OnStart()
```

```

{
    string text="The quick brown fox jumps over the lazy dog";
    string keystr="ABCDEFG";
    uchar src[],dst[],key[];
//--- подготовка ключа шифрования
    StringToCharArray(keystr,key);
//--- подготовка исходного массива src[]
    StringToCharArray(text,src);
//--- вывод исходных данных
    PrintFormat("Initial data: size=%d, string='%s'",ArraySize(src),CharArrayToString(src));
//--- шифрование массива src[] методом DES с 56-битным ключом key[]
    int res=CryptEncode(CRYPT_DES,src,key,dst);
//--- проверка результата шифрования
    if(res>0)
    {
        //--- вывод шифрованных данных
        PrintFormat("Encoded data: size=%d %s",res,ArrayToHex(dst));
        //--- расшифровка данных массива dst[] методом DES с 56-битным ключом key[]
        res=CryptDecode(CRYPT_DES,dst,key,src);
        //--- проверка результата
        if(res>0)
        {
            //--- вывод дешифрованных данных
            PrintFormat("Decoded data: size=%d, string='%s'",ArraySize(src),CharArrayToString(src));
        }
        else
            Print("Ошибка в CryptDecode. Код ошибки=",GetLastError());
    }
    else
        Print("Ошибка в CryptEncode. Код ошибки=",GetLastError());
}

```

#### Смотри также

[Операции с массивами](#), [CryptDecode\(\)](#)

## CryptDecode

Производит обратное преобразование данных массива, полученного при помощи функции [CryptEncode\(\)](#).

```
int CryptDecode(
    ENUM_CRYPT_METHOD method,           // метод преобразования
    const uchar&      data[],          // массив-источник
    const uchar&      key[],           // ключ шифрования
    uchar&            result[]         // массив-приемник
);
```

### Параметры

*method*

[in] Метод преобразования. Может быть одним из значений перечисления [ENUM\\_CRYPT\\_METHOD](#).

*data []*

[in] Массив-источник.

*key []*

[in] Ключ шифрования.

*result []*

[out] Массив-приемник.

### Возвращаемое значение

Количество байт в массиве-приемнике или 0 в случае ошибки. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Смотри также

[Операции с массивами](#), [CryptEncode\(\)](#)

## DebugBreak

Программная точка останова при отладке.

```
void DebugBreak();
```

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

Прерывание выполнения MQL5-программы происходит только в том случае, если программа запущена в режиме отладки. Функцию можно использовать для просмотра значений переменных и/или дальнейшего пошагового выполнения.

## ExpertRemove

Прекращает работу [эксперта](#) и выгружает его с графика.

```
void ExpertRemove();
```

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

Остановка эксперта не происходит немедленно при вызове функции ExpertRemove(), производится лишь взвод флага для прекращения работы эксперта. Т.е., любое следующее событие эксперт обрабатывать уже не будет, произойдет вызов [OnDeinit\(\)](#) и выгрузка с удалением с графика.

Вызов [ExpertRemove\(\)](#) в тестере стратегий внутри обработчика [OnInit\(\)](#) приведет к отмене тестирования на текущем наборе параметров. Такое завершение рассматривается как ошибка при инициализации.

При вызове [ExpertRemove\(\)](#) в тестере стратегий после [успешной инициализации](#) советника тестирование завершится штатным образом с вызовом [OnDeinit\(\)](#) и [OnTester\(\)](#). В этом случае будет получена вся торговая статистика и значение [критерия оптимизации](#), но при этом сам советник будет выгружен из памяти [агента тестирования](#). Это означает, что для тестирования следующего прохода оптимизации агенту понадобится время на повторную загрузку советника. Поэтому для досрочного штатного завершения тестирования использование TesterStop() является наиболее предпочтительным вариантом.

### Пример:

```
//+-----+
//|                                         Test_ExpertRemove.mq5 |
//|                                         Copyright 2009, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
input int ticks_to_close=20; // количество тиков до снятия эксперта
//+-----+
//| Expert deinitialization function          |
//+-----+
void OnDeinit(const int reason)
{
//---
    Print(TimeCurrent(),":", __FUNCTION__," reason code =",reason);
//--- "clear" comment
    Comment("");
//---
}
```

```
//| Expert tick function
//+-----+
void OnTick()
{
    static int tick_counter=0;
//---

    tick_counter++;
    Comment("\nДо выгрузки эксперта ",__FILE__," осталось ",
           (ticks_to_close-tick_counter)," тиков ");
//--- до
    if(tick_counter>=ticks_to_close)
    {
        ExpertRemove();
        Print(TimeCurrent(),":",__FUNCTION__," эксперт будет выгружен");
    }
    Print("tick_counter = ",tick_counter);
//---
}
```

#### Смотри также

[Выполнение программ](#), [События клиентского терминала](#)

## GetPointer

Возвращает указатель объекта.

```
void* GetPointer(
    any_class anyobject // объект любого класса
);
```

### Параметры

*anyobject*  
[in] Объект любого класса.

### Возвращаемое значение

Возвращает указатель объекта.

### Примечание

Только объекты классов имеют указатели. Экземпляры [структур](#) и переменные простых типов указателей не имеют. Объект класса, который не создан при помощи оператора `new()`, а например, автоматически созданный в массиве объектов, все равно имеет указатель. Только этот указатель будет иметь автоматический тип `PINTER_AUTOMATIC`, и к нему нельзя применить оператор [delete\(\)](#). В остальном указатель типа ничем не отличается от динамических указателей типа [PINTER\\_AUTOMATIC](#).

Так как переменные типа структур и простых типов не имеют указателей, то применять к ним функцию `GetPointer()` запрещено. Также запрещается в качестве аргумента функции передавать указатель. Во всех перечисленных случаях компилятор сообщит об ошибке.

Попытка к обращения к некорректному указателю приводит к [критическому завершению](#) программы. Поэтому существует необходимость использования функции [CheckPointer\(\)](#) перед использованием указателя. Указатель может быть некорректным в следующих случаях:

- указатель равен [NULL](#);
- если объект был уничтожен при помощи оператора [delete](#).

Данную функцию можно использовать как проверку указателя на корректность. Значение, отличное от нуля, гарантирует, что по этому указателю можно получить доступ к данным.

### Пример:

```
//+-----+
//| Check_GetPointer.mq5 |
//| Copyright 2009, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

//+-----+
//| Класс, реализующий элемент списка |
//+-----+
```

```

class CItem
{
    int             m_id;
    string          m_comment;
    CItem*          m_next;

public:
    CItem() { m_id=0; m_comment=NULL; m_next=NULL; }
    ~CItem() { Print("Destructor of ",m_id,
                      (CheckPointer(GetPointer(this))==POINTER_DYNAMIC)
                      "dynamic":"non-dynamic")); }

    void Initialize(int id,string comm) { m_id=id; m_comment=comm; }
    void PrintMe() { Print(__FUNCTION__,":",m_id,m_comment); }
    int Identifier() { return(m_id); }
    CItem* Next() { return(m_next); }
    void Next(CItem *item) { m_next=item; }

};

//+-----+
// | Простейший класс списка
//+-----+

class CMyList
{
    CItem*          m_items;

public:
    CMyList() { m_items=NULL; }
    ~CMyList() { Destroy(); }

    bool InsertToBegin(CItem* item);
    void Destroy();

};

//+-----+
// | Вставляет элемент списка в самое начало
//+-----+

bool CMyList::InsertToBegin(CItem* item)
{
    if(CheckPointer(item)==POINTER_INVALID) return(false);

    item.Next(m_items);
    m_items=item;

    return(true);
}

//+-----+
// | Уничтожает список через уничтожение элементов
//+-----+

void CMyList::Destroy()
{
    //--- служебный указатель для работы в цикле
    CItem* item;
    //--- пройдемся в цикле и попытаемся удалить динамические указатели
    while(CheckPointer(m_items)!=POINTER_INVALID)
}

```

```

{
    item=m_items;
    m_items=m_items.Next();
    if(CheckPointer(item)==POINTER_DYNAMIC)
    {
        Print("Dynamyc object",item.Identifier(),"to be deleted");
        delete (item);
    }
    else Print("Non-dynamic object",item.Identifier(),"cannot be deleted");
}
//---
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
    CMyList list;
    CItem items[10];
    CItem* item;
//--- создадим и добавим в список динамический указатель объекта
    item=new CItem;
    if(item!=NULL)
    {
        item.Initialize(100,"dynamic");
        item.PrintMe();
        list.InsertToBegin(item);
    }
//--- добавим автоматические указатели в список
    for(int i=0; i<10; i++)
    {
        items[i].Initialize(i,"automatic");
        items[i].PrintMe();
        item=GetPointer(items[i]);
        if(CheckPointer(item)!=POINTER_INVALID)
            list.InsertToBegin(item);
    }
//--- добавим еще один динамический указатель объекта в начало списка
    item=new CItem;
    if(item!=NULL)
    {
        item.Initialize(200,"dynamic");
        item.PrintMe();
        list.InsertToBegin(item);
    }
//--- удалим элементы списка
    list.Destroy();
//--- все элементы списка будут уничтожены,
//--- смотри в терминале закладку Experts
}

```

{}

**Смотри также**[Указатели объектов](#), [Проверка указателя объекта](#), [Оператор уничтожения объекта delete](#)

## GetTickCount

Функция GetTickCount() возвращает количество миллисекунд, прошедших с момента старта системы.

```
uint GetTickCount();
```

### Возвращаемое значение

Значение типа uint.

### Примечание

Счетчик ограничен разрешающей способностью системного таймера. Так как время хранится как беззнаковое целое, то он переполняется каждые 49.7 дней при непрерывной работе компьютера.

### Пример:

```
#define MAX_SIZE 40
//+-----+
//| скрипт для замера времени вычисления 40 чисел Фибоначчи |
//+-----+
void OnStart()
{
//--- запомним начальное значение
    uint start=GetTickCount();
//--- переменная для получения очередного числа из ряда Фибоначчи
    long fib=0;
//--- цикл, в котором вычисляем заданное количество чисел из ряда Фибоначчи
    for(int i=0;i<MAX_SIZE;i++)
        fib=TestFibo(i);
//--- получим затраченное время в миллисекундах
    uint time=GetTickCount()-start;
//--- выведем в журнал "Эксперты" сообщение
    PrintFormat("Вычисление %d первых чисел Фибоначчи заняло %d ms",MAX_SIZE,time);
//--- работа скрипта завершена
    return;
}
//+-----+
//| Функция получения числа Фибоначчи по его порядковому номеру |
//+-----+
long TestFibo(long n)
{
//--- первый член ряда Фибоначчи
    if(n<2) return(1);
//--- все последующие члены вычисляются по этой формуле
    return(TestFibo(n-2)+TestFibo(n-1));
}
```

### Смотри также

[Дата и время, EventSetMillisecondTimer](#)

## GetMicrosecondCount

Функция GetMicrosecondCount() возвращает количество микросекунд, прошедших с момента начала работы MQL5-программы.

```
ulong GetMicrosecondCount();
```

### Возвращаемое значение

Значение типа ulong.

### Пример:

```
//+-----+
//| Тестируемый код
//+-----+
void Test()
{
    int    res_int=0;
    double res_double=0;
//---
    for(int i=0;i<10000;i++)
    {
        res_int+=i*i;
        res_int++;
        res_double+=i*i;
        res_double++;
    }
}
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    uint    ui=0,ui_max=0,ui_min=INT_MAX;
    ulong   ul=0,ul_max=0,ul_min=INT_MAX;
//--- количество тестов
    for(int count=0;count<1000;count++)
    {
        uint  ui_res=0;
        ulong ul_res=0;
//---
        for(int n=0;n<2;n++)
        {
            //--- выбираем способ измерения
            if(n==0)
                ui=GetTickCount();
            else
                ul=GetMicrosecondCount();
//--- тестируемый код
    }
}
```

```
Test();
//--- копим результат измерения (в зависимости от способа)
if(n==0)
    ui_res+=GetTickCount()-ui;
else
    ul_res+=GetMicrosecondCount()-ul;
}

//--- собираем минимальное и максимальное время исполнения кода обоих измерений
if(ui_min>ui_res)
    ui_min=ui_res;
if(ui_max<ui_res)
    ui_max=ui_res;
if(ul_min>ul_res)
    ul_min=ul_res;
if(ul_max<ul_res)
    ul_max=ul_res;
}

//---
Print("GetTickCount error (msec) : ",ui_max-ui_min);
Print("GetMicrosecondCount error (msec) : ",DoubleToString((ul_max-ul_min)/1000.0,2))
}
```

#### Смотри также

[Дата и время](#)

## MessageBox

Создает и отображает окно сообщений, а также управляет им. Окно сообщений содержит сообщение и заголовок, любую комбинацию предопределенных значков и командных кнопок.

```
int MessageBox(
    string text,           // текст сообщения
    string caption=NULL,   // заголовок окна
    int flags=0            // определяет набор кнопок в окне
);
```

### Параметры

*text*

[in] Текст, содержащий сообщение для отображения.

*caption=NULL*

[in] Необязательный текст для отображения в заголовке окна сообщения. Если этот параметр пустой, в заголовке окна будет отображено название эксперта.

*flags=0*

[in] Необязательные [флаги](#), определяющие вид и поведение диалогового окна. Флаги могут быть комбинацией специальной группы флагов .

### Возвращаемое значение

Если функция успешно выполняется, возвращаемое значение - одно из значений кодов возврата [MessageBox\(\)](#).

### Примечание

Функцию не рекомендуется использовать в пользовательских индикаторах, так как вызов `MessageBox()` приостанавливает работу [потока исполнения](#) индикатора на всё время ожидания ответа пользователя. А так как все индикаторы по каждому символу выполняются в едином потоке, то будут остановлены все графики на всех таймфреймах по данному символу.

При работе в [тестере стратегий](#) функция `MessageBox()` не выполняется.

## PeriodSeconds

Возвращает количество секунд в периоде.

```
int PeriodSeconds(
    ENUM_TIMEFRAMES period=PERIOD_CURRENT           // период графика
);
```

### Параметры

*period=PERIOD\_CURRENT*

[in] Значение периода графика из перечисления [ENUM\\_TIMEFRAMES](#). Если параметр не указан, то возвращается количество секунд текущего периода графика, на котором запущена программа.

### Возвращаемое значение

Количество секунд в указанном периоде.

### Смотри также

[Period](#), [Периоды графиков](#), [Дата и время](#), [Видимость объектов](#)

## PlaySound

Воспроизводит звуковой файл.

```
bool PlaySound(  
    string filename // имя файла  
) ;
```

### Параметры

*filename*

[in] Путь к звуковому файлу. Если *filename*=NULL, воспроизведение звука прекращается.

### Возвращаемое значение

true - если звуковой файл найден, иначе возвращает false.

### Примечание

Файл должен быть расположен в каталоге каталог\_терминала\Sounds или его подкаталоге. Проигрываются только звуковые файлы в формате WAV.

Вызов PlaySound() с параметром NULL останавливает воспроизведение звука.

При работе в [тестере стратегий](#) функция PlaySound() не выполняется.

### Смотри также

[Ресурсы](#)

## Print

Печатает некоторое сообщение в журнал экспертов. Параметры могут иметь любой тип.

```
void Print(
    argument,      // первое значение
    ...
    // последующие значения
);
```

### Параметры

...

[in] Любые значения, разделенные запятыми. Количество параметров не может превышать 64.

### Примечание

Массивы нельзя передавать в функцию Print(). Массивы должны печататься поэлементно.

Данные типа double выводятся с точностью до 16 десятичных цифр после точки, при этом данные могут выводиться либо в традиционном либо в научном формате - в зависимости от того, как запись будет наиболее компактна. Данные типа float выводятся с 5 десятичными цифрами после точки. Для вывода вещественных чисел с другой точностью либо в явно указанном формате необходимо использовать функцию [PrintFormat\(\)](#).

Данные типа bool выводятся в виде строк "true" или "false". Даты выводятся в виде YYYY.MM.DD HH:MI:SS. Для вывода даты в другом формате необходимо использовать функцию [TimeToString\(\)](#). Данные типа color выводятся либо в виде строки R,G,B, либо в виде названия цвета, если этот цвет присутствует в наборе цветов.

При работе в [тестере стратегий](#) в режиме оптимизации функция Print() не выполняется.

### Пример:

```
void OnStart()
{
//--- выведем DBL_MAX с помощью Print(), это равносильно PrintFormat(%.16G,DBL_MAX)
Print("---- как выглядит DBL_MAX ----");
Print("Print(DBL_MAX)=",DBL_MAX);
//--- теперь выведем число DBL_MAX с помощью PrintFormat()
PrintFormat("PrintFormat(%.16G,DBL_MAX)=%.16G",DBL_MAX);
//--- Вывод в журнал "Эксперты"
// Print(DBL_MAX)=1.797693134862316e+308
// PrintFormat(%.16G,DBL_MAX)=1.797693134862316E+308

//--- посмотрим как выводится тип float
float c=(float)M_PI; // нужно явно приводить к целевому типу
Print("c=",c, " Pi=",M_PI, " (float)M_PI=", (float)M_PI);
// c=3.14159 Pi=3.141592653589793 (float)M_PI=3.14159

//--- покажем, что может произойти при арифметических операциях над вещественными типами
double a=7,b=200;
Print("---- перед арифметическими операциями");
```

```

Print("a=",a,"    b=",b);
Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- разделим a на b (7/200)
a=a/b;
//--- теперь как будто восстановим значение в переменной b
b=7.0/a; // ожидается, что b=7.0/(7.0/200.0)=>7.0/7.0*200.0=200 - но это не так
//--- выведем вновь вычисленное значение b
Print("----- после арифметических операций");
Print("Print(b)=",b);
Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- вывод в журнал "Эксперты"
// Print(b)=200.0
// Print(DoubleToString(b,16))=199.99999999999716 (видим, что на самом деле b уже не
//--- создадим очень маленькое значение epsilon=1E-013
double epsilon=1e-13;
Print("----- создадим очень маленькое число");
Print("epsilon=",epsilon); // получим epsilon=1E-013
//--- теперь вычтем эпсилон из числа b и выведем снова значение в журнал "Эксперты"
b=b-epsilon;
//--- выводим двумя способами
Print("----- после вычитания epsilon из переменной b");
Print("Print(b)=",b);
Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- вывод в журнал "Эксперты"
// Print(b)=199.999999999999 (теперь значение b после вычитания эпсилон не может окр
// Print(DoubleToString(b,16))=199.999999999998578
//      (теперь значение b после вычитания эпсилон не может округлиться до 200)
}

```

#### Смотри также

[DoubleToString](#), [StringFormat](#)

## PrintFormat

Форматирует и печатает наборы символов и значений в журнал экспертов в соответствии с заданным форматом.

```
void PrintFormat(
    string format_string, // форматная строка
    ...
    // значения простых типов
);
```

### Параметры

*format\_string*

[in] Стока формата состоит из обычных символов и, если за строкой формата следуют аргументы, еще и спецификации формата.

...

[in] Любые значения простых типов, разделенные запятыми. Общее количество параметров не может превышать 64, включая форматную строку.

### Возвращаемое значение

Строка.

### Примечание

При работе в [тестере стратегий](#) в режиме оптимизации функция PrintFormat() не выполняется.

Количество, порядок и тип параметров должны точно соответствовать составу спецификаторов, в противном случае результат печати неопределён. Вместо функции PrintFormat() можно использовать функцию [printf\(\)](#).

Если за строкой формата следуют еще параметры, то эта строка должна содержать спецификации формата, определяющие формат вывода этих параметров. Спецификация формата всегда начинается с символа знака процента (%).

Строка формата читается слева направо. Когда встречается первая спецификация формата (если она есть), то значение первого параметра после строки формата преобразовывается и выводится согласно заданной спецификации. Вторая спецификация формата вызывает преобразование и вывод второго параметра и так далее, до конца строки формата.

Спецификация формата имеет следующую форму:

**%[flags][width].[precision]{[h | l | ll | I32 | I64]}type**

Каждое поле форматной спецификации является либо простым символом, либо числом, обозначающим обычную форматную опцию. Простейшая спецификация формата содержит только знак процента (%) и символ, определяющий [тип выводимого параметра](#) (например %s). Если требуется в форматной строке вывести символ знака процента, то необходимо использовать форматную спецификацию %%.

### flags

Флаг	Описание	Поведение по умолчанию
- (минус)	Выравнивание по левому краю в пределах заданной ширины	Выравнивание по правому краю
+ (плюс)	Вывод знака + или - для значений знаковых типов	Знак выводится только если значение отрицательное
0 (ноль)	Перед выводимом значением добавляются нули в пределах заданной <a href="#">ширины</a> . Если указан флаг 0 с целочисленным форматом (i, u, x, X, o, d) и задана спецификация точности (например, %04.d), то 0 игнорируется.	Ничего не вставляется
пробел	Перед выводимым значением ставится пробел, если значение является знаковым и положительным	Пробелы не вставляются
#	Если используется совместно с форматом o, x или X, то перед выводимым значением добавляется 0, 0x или 0X соответственно.	Ничего не вставляется
	Если используется совместно с форматом e, E, a или A, то значение всегда выводится с десятичной точкой.	Десятичная точка выводится только если есть ненулевая дробная часть.
	Если используется совместно с форматом g или G, флаг определяет наличие десятичной точки в выводимом значении и препятствует отсечению ведущих нулей. Флаг # игнорируется при совместном использовании с форматами c, d, i, u, s.	Десятичная точка выводится только если есть ненулевая дробная часть. Ведущие нули отсекаются

## width

Неотрицательное десятичное число, которое задает минимальное число выводимых символов отформатированного значения. Если количество выводимых символов меньше указанной ширины, то добавляется соответствующее количество пробелов слева или справа в зависимости

от выравнивания (флаг `-`). При наличии флага ноль (0), перед выводимым значением добавляется соответствующее количество нулей. Если число выводимых символов больше заданной ширины, то выводимое значение никогда не усекается.

Если в качестве ширины указана звездочка (\*), то в списке передаваемых параметров на соответствующем месте должно быть значение типа `int`, которое будет использовано для указания ширины выводимого значения.

## precision

Неотрицательное десятичное число, которое определяет точность вывода - количество цифр после десятичной точки. В отличие от спецификации ширины, спецификация точности может отсекать часть дробного значения с округлением или без округления.

Для разных форматных [типов](#) (`type`) спецификация точности применяется по-разному.

Типы	Описание	Поведение по умолчанию
<code>a, A</code>	Спецификация точности указывает количество знаков после десятичной точки	Точность по умолчанию - 6.
<code>c, C</code>	Не применяется	
<code>d, i, u, o, x, X</code>	Указывает минимальное число выводимых цифр. Если количество цифр в соответствующем параметре меньше указанной точности, то выводимое значение дополняется слева нулями. Выводимое значение не обрезается, если количество выводимых цифр больше указанной точности	Точность по умолчанию - 1.
<code>e, E, f</code>	Указывает число выводимых цифр после десятичной точки. Последняя выводимая цифра округляется	Точность по умолчанию - 6. Если указана точность 0 или дробная часть отсутствует, то десятичная точка не выводится.
<code>g, G</code>	Указывает максимальное число значимых цифр	Выводится 6 значимых цифр.
<code>s, S</code>	Указывает количество выводимых символов строки. Если длина строки превышает значение точности, то строка усекается на выводе	Выводится вся строка

**h | l | ll | I32 | I64**

Спецификации размеров данных, передаваемых в качестве параметра.

Тип параметра	Используемый префикс	Совместный спецификатор типа
int	l (маленькая L)	d, i, o, x, or X
uint	l (маленькая L)	o, u, x, or X
long	ll (две маленькие L)	d, i, o, x, or X
short	h	d, i, o, x, or X
ushort	h	o, u, x, or X
int	I32	d, i, o, x, or X
uint	I32	o, u, x, or X
long	I64	d, i, o, x, or X
ulong	I64	o, u, x, or X

**type**

Спецификатор типа является единственным обязательным полем для форматированного вывода.

Символ	Тип	Выводной формат
c	int	Символ типа short (Unicode)
C	int	Символ типа char (ANSI)
d	int	Знаковое десятичное целое
i	int	Знаковое десятичное целое
o	int	Беззнаковое восьмеричное целое
u	int	Беззнаковое десятичное целое
x	int	Беззнаковое шестнадцатиричное целое с использованием "abcdef"
X	int	Беззнаковое шестнадцатиричное целое с использованием "ABCDEF"

e	double	Вещественное значение в формате [ - ]d.dddd e [sign] ddd, где d - одна десятичная цифра, dddd - одна или больше десятичных цифр, ddd - трехзначное число, определяющее размер экспоненты, sign - знак плюс или минус
E	double	Идентично формату e, за исключением того, что знак экспоненты вывоится большой буквой (E вместо e)
f	double	Вещественное значение в формате [ - ]dddd.dddd, где dddd - одна или больше десятичных цифр. Количество выводимых знаков перед десятичной точкой зависит от величины значения числа. Количество знаков после десятичной точки зависит от требуемой точности.
g	double	Вещественное значение, выводимое в формате f или e, в зависимости от того какой вывод будет компактнее.
G	double	Вещественное значение, выводимое в формате f или E, в зависимости от того какой вывод будет компактнее.
a	double	Вещественное значение в формате [-]0xh.hhhh p±dd, где h.hhhh - мантисса в виде шестнадцатиричных цифр с использованием "abcdef", dd - одна или более цифр экспоненты. Количество знаков после запятой определяется <u>спецификацией точности</u>
A	double	Вещественное значение в формате [-]0xh.hhhh P±dd, где h.hhhh - мантисса в виде

		шестнадцатиричных цифр с использованием "ABCDEF", dd - одна или более цифр экспоненты. Количество знаков после запятой определяется <a href="#">спецификацией точности</a>
s	string	Вывод строки

Вместо функции PrintFormat() можно использовать функцию [printf\(\)](#).

**Пример:**

```

void OnStart()
{
//--- имя торгового сервера
    string server=AccountInfoString(ACCOUNT_SERVER);
//--- номер торгового счета
    int login=(int)AccountInfoInteger(ACCOUNT_LOGIN);
//--- вывод значения long
    long leverage=AccountInfoInteger(ACCOUNT_LEVERAGE);
    PrintFormat("%s %d: плечо = 1:%I64d",
               server,login,leverage);
//--- валюта депозита
    string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- вывод значения double с 2 цифрами после десятичной точки
    double equity=AccountInfoDouble(ACCOUNT_EQUITY);
    PrintFormat("%s %d: размер собственных средств на счете = %.2f %s",
               server,login,equity,currency);
//--- вывод значения double с обязательным выводом знака +-
    double profit=AccountInfoDouble(ACCOUNT_PROFIT);
    PrintFormat("%s %d: текущий результат по открытым позициям = %+.2f %s",
               server,login,profit,currency);
//--- вывод значения double с переменным количеством цифр после десятичной точки
    double point_value=SymbolInfoDouble(_Symbol,SYMBOL_POINT);
    string format_string=StringFormat("%%s: значение одного пункта = %%.%df",_Digits);
    PrintFormat(format_string,_Symbol,point_value);
//--- вывод значения int
    int spread=(int)SymbolInfoInteger(_Symbol,SYMBOL_SPREAD);
    PrintFormat("%s: текущий спред в пунктах = %d ",
               _Symbol,spread);
//--- вывод значения double в научном формате с плавающей запятой и точностью 17 знача
    PrintFormat("DBL_MAX = %.17e",DBL_MAX);
//--- вывод значения double в научном формате с плавающей запятой и точностью 17 знача
    PrintFormat("EMPTY_VALUE = %.17e",EMPTY_VALUE);
//--- вывод через PrintFormat() с точностью по умолчанию
    PrintFormat("PrintFormat(EMPTY_VALUE) = %e",EMPTY_VALUE);
//--- простой вывод через Print()
    Print("Print(EMPTY_VALUE) = ",EMPTY_VALUE);
/* результат выполнения
MetaQuotes-Demo 1889998: плечо = 1:100
MetaQuotes-Demo 1889998: размер собственных средств на счете = 22139.86 USD
MetaQuotes-Demo 1889998: текущий результат по открытым позициям = +174.00 USD
EURUSD: значение одного пункта = 0.00001
EURUSD: текущий спред в пунктах = 12
DBL_MAX = 1.79769313486231570e+308
EMPTY_VALUE = 1.79769313486231570e+308
PrintFormat(EMPTY_VALUE) = 1.797693e+308
Print(EMPTY_VALUE) = 1.797693134862316e+308
*/
}

```

### Смотри также

[StringFormat](#), [DoubleToString](#), [Вещественные типы \(double, float\)](#)

## ResetLastError

Устанавливает значение предопределенной переменной [\\_LastError](#) в ноль.

```
void ResetLastError();
```

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

Необходимо отметить, что функция [GetLastError\(\)](#) не обнуляет переменную [\\_LastError](#). Обычно вызов функции `ResetLastError()` производится перед вызовом функции, после которой проверяется возникновение [ошибки](#).

## ResourceCreate

Создает ресурс изображения на основе набора данных. Существует два варианта функции:

### Создание ресурса на основе файла

```
bool ResourceCreate(
    const string      resource_name,           // имя ресурса
    const string      path                      // относительный путь к файлу
);
```

### Динамическое создание ресурса на основе массива пикселей

```
bool ResourceCreate(
    const string      resource_name,           // имя ресурса
    const uint&       data[],                  // набор данных в виде массива
    uint              img_width,               // ширина создаваемой картинки-ресурса
    uint              img_height,              // высота создаваемой картинки-ресурса
    uint              data_xoffset,             // смещение левого верхнего угла создаваемой
    uint              data_yoffset,             // смещение левого верхнего угла создаваемой
    uint              data_width,               // общая ширина изображения на основе набора
    ENUM_COLOR_FORMAT color_format            // способ обработки цвета
);
```

### Параметры

*resource\_name*

[in] Имя ресурса.

*path*

[in] Относительный путь к файлу, содержащему данные для ресурса. Если путь начинается с обратной косой черты "\", (пишется "\\\"), то файл ищется относительно папки каталог\_данных\_терминала\MQL5\. Если обратной косой черты нет, то ресурс ищется относительно расположения EX5-файла, из которого вызывается функция.

*data[][][]*

[in] Одномерный или двумерный массив для создания полного изображения.

*img\_width*

[in] Ширина прямоугольной области изображения в пикселях для помещения в ресурс в виде картинки. Не может быть больше значения *data\_width*.

*img\_height*

[in] Высота прямоугольной области изображения в пикселях для помещения в ресурс в виде картинки.

*data\_xoffset*

[in] Смещение в пикселях прямоугольной области изображения по горизонтали вправо.

*data\_yoffset*

[in] Смещение в пикселях прямоугольной области изображения по вертикали вниз.

*data\_width*

[in] Требуется только для одномерных массивов и означает полную ширину создаваемого изображения из набора данных. Если *data\_width*=0, то подразумевается равным *img\_width*. Для двумерных массивов данный параметр игнорируется и принимается равным второй размерности массива *data[]*.

*color\_format*

[in] Способ обработки цвета из перечисления [ENUM\\_COLOR\\_FORMAT](#).

#### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4015 - ERR\_RESOURCE\_NAME\_DUPLICATED (совпадение имен динамического и [статического](#) ресурсов),
- 4016 - ERR\_RESOURCE\_NOT\_FOUND (ресурс не найден),
- 4017 - ERR\_RESOURCE\_UNSUPPORTED\_TYPE (тип ресурса не поддерживается),
- 4018 - ERR\_RESOURCE\_NAME\_IS\_TOO\_LONG (слишком длинное имя ресурса).

#### Примечание

Если второй вариант функции вызывается для создания одного и того же ресурса с разными параметрами ширины, высоты и сдвига, то новый ресурс не пересоздается, а просто обновляется существующий.

Первый вариант функции позволяет загружать из файлов картинки и звуки, второй вариант предназначен только для динамического создания изображений.

Картинки должны быть в формате BMP с глубиной цвета 24 или 32 бита, звуки могут быть только в формате WAV. Размер ресурса не должен превышать 16 Mb.

#### ENUM\_COLOR\_FORMAT

Идентификатор	Описание
COLOR_FORMAT_XRGB_NOALPHA	Компонента альфа-канала игнорируется
COLOR_FORMAT_ARGB_RAW	Компоненты цвета не обрабатываются терминалом (должны быть корректно заданы пользователем)
COLOR_FORMAT_ARGB_NORMALIZE	Компоненты цвета обрабатываются терминалом

#### Смотри также

[Ресурсы](#), [ObjectCreate\(\)](#), [ObjectSetString\(\)](#), [OBJPROP\\_BMPFILE](#)

## ResourceFree

Удаляет [динамически созданный ресурс](#) (освобождает занятую ресурсом память).

```
bool ResourceFree(
    const string resource_name           // имя ресурса
);
```

### Параметры

*resource\_name*  
[in] Имя [ресурса](#), должно начинаться с "::".

### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция ResourceFree() позволяет разработчику MQL5-программы управлять потреблением памяти при активной работе с ресурсами. [Графические объекты](#), привязанные к удаляемому из памяти ресурсу, будут отображаться правильно и после его удаления. Но вновь созданные графические объекты ([OBJ\\_BITMAP](#) и [OBJ\\_BITMAP\\_LABEL](#)) уже не смогут использовать удалённый ресурс.

Функция удаляет только динамические ресурсы, созданные данной программой.

### Смотри также

[Ресурсы](#), [ObjectCreate\(\)](#), [PlaySound\(\)](#), [ObjectSetString\(\)](#), [OBJPROP\\_BMPFILE](#)

## ResourceReadImage

Читает данные графического ресурса, [созданного функцией ResourceCreate\(\)](#) или [сохраненного в EX5-файле при компиляции](#).

```
bool ResourceReadImage(
    const string      resource_name,           // имя графического ресурса для чтения
    uint&             data[],                 // массив для получения данных из ресурса
    uint&             width,                  // для получения ширины картинки в ресурсе
    uint&             height,                 // для получения высоты картинки в ресурсе
);
```

### Параметры

*resource\_name*

[in] Имя графического ресурса, содержащего изображение. Для доступа к собственным ресурсам указывается в коротком виде ">::resourcename". Если же необходимо загрузить ресурс из скомпилированного EX5-файла, то необходимо имя в полном виде с указанием пути относительно папки MQL5, имени файла и имени ресурса - "path\\filename.ex5::resourcename".

*data[] []*

[in] Одномерный или двумерный массив для получения данных из графического ресурса.

*img\_width*

[out] Ширина картинки графического ресурса в пикселях .

*img\_height*

[out] Высота картинки графического ресурса в пикселях .

### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Если на основании массива *data[]* в дальнейшем необходимо [создать графический ресурс](#), то следует использовать [формат цвета](#) COLOR\_FORMAT\_ARGB\_NORMALIZE или COLOR\_FORMAT\_XRGB\_NOALPHA.

Если массив *data[]* является двумерным и его вторая размерность меньше размера X(*width*) графического ресурса, то функция ResourceReadImage() вернет false и чтение не будет произведено. Но при этом, если ресурс существует, то в параметры *width* и *height* возвращаются актуальные размеры картинки. Это позволит сделать еще одну попытку получения данных из ресурса.

### Смотри также

[Ресурсы](#), [ObjectCreate\(\)](#), [ObjectSetString\(\)](#), [OBJPROP\\_BMPFILE](#)

## ResourceSave

Сохраняет ресурс в указанный файл.

```
bool ResourceSave(
    const string resource_name      // имя ресурса
    const string file_name          // имя файла
);
```

### Параметры

*resource\_name*

[in] Имя ресурса, должно начинаться с "::".

*file\_name*

[in] Имя файла относительно MQL5\Files.

### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция всегда перезаписывает файл и создаёт при необходимости все промежуточные подкаталоги в имени файла в случае их отсутствия.

### Смотри также

[Ресурсы](#), [ObjectCreate\(\)](#), [PlaySound\(\)](#), [ObjectSetString\(\)](#), [OBJPROP\\_BMPFILE](#)

## SetReturnError

Устанавливает код возврата, который вернёт процесс терминала при завершение работы.

```
void SetReturnError(  
    int ret_code           // код завершения клиентского терминала  
);
```

### Параметры

*ret\_code*

[in] Код возврата, который будет возвращен процессом клиентского терминала при завершении работы.

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

Установка заданного кода возврата *ret\_code* с помощью функции SetReturnError() будет полезна для анализа причин программного прекращения работы при [запуске терминала из командной строки](#).

Функция SetReturnError() в отличие от [TerminalClose\(\)](#) не завершает работу терминала, она только устанавливает код возврата, который вернёт процесс терминала при его завершении.

Если функция SetReturnError() вызывалась многократно и/или из разных MQL5 программ, то терминал вернёт последний установленный код возврата.

Выставленный код будет возвращаться при завершении процесса терминала кроме следующих случаев:

- возникла [критическая ошибка](#) во время выполнения;
- была вызвана функции TerminalClose(int *ret\_code*), которая отдает терминалу команду на завершение работы с заданным кодом.

### Смотри также

[Выполнение программ](#), [Ошибки выполнения](#), [Причины деинициализации](#), [TerminalClose](#)

## SetUserError

Устанавливает предопределенную переменную [LastError](#) в значение, равное [ERR\\_USER\\_ERROR\\_FIRST](#) + user\_error

```
void SetUserError(  
    ushort user_error, // номер ошибки  
);
```

### Параметры

*user\_error*  
[in] Номер [ошибки](#), устанавливаемый пользователем.

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

После того, как была выставлена ошибка при помощи функции SetUserError(user\_error), функция [GetLastError\(\)](#) вернет значение, равное [ERR\\_USER\\_ERROR\\_FIRST](#) + user\_error.

### Пример:

```
void OnStart()  
{  
//--- установим номер ошибки 65537=(ERR_USER_ERROR_FIRST +1)  
    SetUserError(1);  
//--- получим код последней ошибки  
    Print("GetLastError = ",GetLastError());  
/*  
    Результат  
    GetLastError = 65537  
*/  
}
```

## Sleep

Функция задерживает выполнение текущего эксперта или скрипта на определенный интервал.

```
void Sleep(  
    int milliseconds // интервал  
) ;
```

### Параметры

*milliseconds*

[in] Интервал задержки в миллисекундах.

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

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.

## TerminalClose

Посыпает терминалу команду на завершение работы.

```
bool TerminalClose(
    int ret_code          // код завершения клиентского терминала
);
```

### Параметры

*ret\_code*

[in] Код возврата, возвращаемый процессом клиентского терминала при завершении работы.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Функция TerminalClose() не производит немедленной остановки работы терминала, она просто посыпает терминалу команду на завершение.

В коде советника, вызвавшего TerminalClose(), должны быть сделаны все приготовления для немедленного завершения работы (например, должны быть штатным образом закрыты все ранее открытые файлы). Сразу после вызова этой функции должен идти [оператор return](#).

Параметр *ret\_code* позволяет указывать нужный код возврата для анализа причин программного прекращения работы терминала при его запуске из командной строки.

### Пример:

```
//--- input parameters
input int tiks_before=500; // количество тиков до завершения
input int pips_to_go=15;   // расстояние в пипсах
input int seconds_st=50;   // сколько секунд даем эксперту
//--- globals
datetime launch_time;
int tick_counter=0;
//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//---
    Print(__FUNCTION__," reason code = ",reason);
    Comment("");
}
//+-----+
//| Expert tick function
//+-----+
void OnTick()
{
    static double first_bid=0.0;
```

```

MqlTick      tick;
double       distance;
//---

SymbolInfoTick(_Symbol,tick);
tick_counter++;
if(first_bid==0.0)
{
    launch_time=tick.time;
    first_bid=tick.bid;
    Print("first_bid =",first_bid);
    return;
}
//--- ход цены в пунктах
distance=(tick.bid-first_bid)/_Point;
//--- выведем сообщение, чтобы отслеживать работу советника
string comm="С момента запуска:\r\n\x25CF прошло секунд: "+
    IntegerToString(tick.time-launch_time)+" ;"+
    "\r\n\x25CF поступило тиков: "+(string)tick_counter+" ;"+
    "\r\n\x25CF цена прошла в пунктах: "+StringFormat("%G",distance);
Comment(comm);
//--- секция проверки условий для закрытия терминала
if(tick_counter>=tiks_before)
    TerminalClose(0); // выход по счетчику тиков
if(distance>pips_to_go)
    TerminalClose(1); // прошли вверх на pips_to_go пипсов
if(distance<-pips_to_go)
    TerminalClose(-1); // прошли вниз на pips_to_go пипсов
if(tick.time-launch_time>seconds_st)
    TerminalClose(100); // завершение работы по таймауту
//---
}

```

#### Смотри также

[Выполнение программ](#), [Ошибки выполнения](#), [Причины deinициализации](#)

## TesterHideIndicators

Задает режим показа/скрытия индикаторов, которые используются в эксперте. Функция предназначена для управления видимостью используемых индикаторов только при тестировании.

```
void TesterHideIndicators(
    bool      hide      // флаг
);
```

### Параметры

*hide*

[in] Флаг скрытия индикаторов при тестировании. Укажите **true**, если необходимо скрывать создаваемые индикаторы, иначе **false**.

### Возвращаемое значение

Нет.

### Примечание

По умолчанию на графике визуального тестирования показываются все индикаторы, которые создаются в тестируемом эксперте. Также эти индикаторы показываются на графике, который автоматически открывается по окончании тестирования. Функция `TesterHideIndicators()` позволяет разработчику в коде эксперта прописать запрет на показ используемых индикаторов.

Для того чтобы запретить показ используемого индикатора при тестировании советника, необходимо перед созданием его хенда вызвать `TesterHideIndicators()` с параметром **true** - все создаваемые после этого индикаторы будут помечены флагом скрытия. Индикаторы, помеченные флагом скрытия, не показываются при визуальном тестировании и на графике, который автоматически открывается при завершении тестирования.

Чтобы отключить режим скрытия вновь создаваемых индикаторов, нужно вновь вызвать `TesterHideIndicators()`, но уже с параметром **false**. На графике тестирования могут быть выведены только те индикаторы, которые непосредственно создаются из тестируемого эксперта. Это правило относится только к тем случаям, когда нет ни одного шаблона в папке <каталог\_данных>MQL5\Profiles\Templates.

Если в папке <каталог\_данных>MQL5\Profiles\Templates присутствует специальный шаблон <имя\_эксперта>.tpl, то при визуальном тестировании и на графике тестирования будут показаны только индикаторы из данного шаблона. В этом случае никакие индикаторы, используемые в тестируемом эксперте, показаны не будут. Даже если в коде советника вызывалась функция `TesterHideIndicators()` с параметром **true**.

Если же в папке <каталог\_данных>MQL5\Profiles\Templates нет специального шаблона <имя\_эксперта>.tpl, но есть шаблон tester.tpl, то при визуальном тестировании и на графике тестирования будут показаны индикаторы из шаблона tester.tpl и индикаторы из советника, которые не запрещены к показу с помощью функции `TesterHideIndicators()`. Если шаблона tester.tpl нет, то вместо него будут использованы индикаторы из шаблона default.tpl.

Если тестер стратегий не найдет ни одного подходящего шаблона (<имя\_эксперта>.tpl, tester.tpl или default.tpl), то показ используемых в советнике индикаторов полностью управляет функцией `TesterHideIndicators()`.

Пример:

```
bool CSampleExpert::InitIndicators(void)
{
    TesterHideIndicators(true);

    //--- create MACD indicator
    if(m_handle_macd==INVALID_HANDLE)
        if((m_handle_macd=iMACD(NULL,0,12,26,9,PRICE_CLOSE))==INVALID_HANDLE)
    {
        printf("Error creating MACD indicator");
        return(false);
    }

    TesterHideIndicators(false);

    //--- create EMA indicator and add it to collection
    if(m_handle_ema==INVALID_HANDLE)
        if((m_handle_ema=iMA(NULL,0,InpMATrendPeriod,0,MODE_EMA,PRICE_CLOSE))==INVALID_HANDLE)
    {
        printf("Error creating EMA indicator");
        return(false);
    }

    //--- succeed
    return(true);
}
```

Смотри также

[IndicatorRelease](#)

## TesterStatistics

Возвращает значение указанного статистического показателя, рассчитанного по результатам тестирования

```
double TesterStatistics(  
    ENUM_STATISTICS statistic_id      // идентификатор  
);
```

### Параметры

*statistic\_id*

[in] Идентификатор статистического показателя из перечисления [ENUM\\_STATISTICS](#).

### Возвращаемое значение

Значение статистического показателя из результатов тестирования.

### Примечание

Функция может быть вызвана внутри [OnTester\(\)](#) или [OnDeinit\(\)](#) в тестере. В других случаях результат неопределён.

## TesterStop

Отдает команду на завершении работы программы при [тестировании](#).

```
void TesterStop();
```

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

Функция TesterStop() предназначена для штатного досрочного завершения работы советника на [агенте тестирования](#) - например, при достижении заданного количества убыточных сделок или заданного уровня просадки.

Вызов TesterStop() считается нормальным завершением тестирования, и поэтому будет вызвана функция [OnTester\(\)](#) с отдачей тестеру стратегий всей накопленной торговой статистики и значения [критерия оптимизации](#).

Вызов [ExpertRemove\(\)](#) в тестере стратегий также означает нормальное завершение тестирования и позволяет получить торговую статистику, но при этом советник выгружается из памяти агента. В этом случае для выполнения прохода на следующем наборе параметров понадобится время на повторную загрузку программы. Поэтому для досрочного штатного завершения тестирования использование TesterStop() является наиболее предпочтительным вариантом.

### Смотри также

[Выполнение программ](#), [Тестирование торговых стратегий](#), [ExpertRemove](#), [SetReturnError](#)

## TesterDeposit

Специальная функция для эмуляции операций внесения средств в процессе тестирования. Может быть использована в некоторых системах управления капиталом.

```
bool TesterDeposit(  
    double money           // размер вносимой суммы  
);
```

### Параметры

*money*

[in] Размер денежных средств, которые необходимо внести на счет в валюте депозита.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Смотри также

[TesterWithdrawal](#)

## TesterWithdrawal

Специальная функция для эмуляции операций снятия средств в процессе тестирования. Может быть использована в некоторых системах управления капиталом.

```
bool TesterWithdrawal(
    double money           // размер снимаемой суммы
);
```

### Параметры

*money*

[in] Размер денежных средств, которые необходимо снять со счета (в валюте депозита).

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Смотри также

[TesterWithdrawal](#)

## TranslateKey

Возвращает Unicode-символ по виртуальному коду клавиши, учитывая текущий язык ввода и состояние управляющих клавиш.

```
short TranslateKey(
    int key_code           // код клавиши для получения Unicode-символа
);
```

### Параметры

*key\_code*  
[in] Код клавиши.

### Возвращаемое значение

Юникодный символ в случае успешного преобразования. В случае ошибки функция вернёт -1.

### Примечание

Функция использует [ToUnicodeEx](#) для преобразования нажатых пользователем клавиш в Unicode-символы. Ошибка может возникнуть в том случае, если не сработает ToUnicodeEx - например, при попытке получить символ для клавиши SHIFT.

### Пример:

```
void OnChartEvent(const int id,const long& lparam,const double& dparam,const string& s)
{
    if(id==CHARTEVENT_KEYDOWN)
    {
        short sym=TranslateKey((int)lparam);
        //--- если введённый символ успешно преобразован в Юникод
        if(sym>0)
            Print(sym,"'",ShortToString(sym),"'");
        else
            Print("Error in TranslateKey for key=",lparam);
    }
}
```

### Смотри также

[События клиентского терминала, OnChartEvent](#)

## ZeroMemory

Функция обнуляет переменную, переданную ей по ссылке.

```
void ZeroMemory(  
    void & variable // обнуляемая переменная  
) ;
```

### Параметры

*variable*

[in] [out] Переменная, передаваемая по ссылке, которую требуется обнулить (инициализировать нулевыми значениями).

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

Если параметром функции является строка, то данный вызов будет эквивалентен указанию для нее значения NULL.

Для простых типов и их массивов, а также структур/классов, состоящих из таких типов, это простое обнуление.

Для объектов, содержащих строки и динамические массивы, производится вызов ZeroMemory() для каждого члена.

Для любых массивов, не защищенных модификатором const, производится обнуление всех элементов.

Для массивов сложных объектов происходит вызов ZeroMemory() для каждого элемента.

Функция ZeroMemory() не применима для классов с защищенными [членами](#) или [наследованием](#).

## 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]`.

Функция	Действие
<a href="#">ArrayBsearch</a>	Возвращает индекс первого найденного элемента в первом измерении массива
<a href="#">ArrayCopy</a>	Копирует один массив в другой
<a href="#">ArrayCompare</a>	Возвращает результат сравнения двух массивов <a href="#">простых типов</a> или пользовательских структур, не имеющих <a href="#">сложных объектов</a>
<a href="#">ArrayFree</a>	Освобождает буфер любого динамического массива и устанавливает размер нулевого измерения в 0 (ноль)
<a href="#">ArrayGetAsSeries</a>	Проверяет направление индексации массива
<a href="#">ArrayInitialize</a>	Устанавливает все элементы числового массива в одну величину
<a href="#">ArrayFill</a>	Заполняет числовой массив указанным значением
<a href="#">ArrayIsSeries</a>	Проверяет, является ли массив таймсерией
<a href="#">ArrayIsDynamic</a>	Проверяет, является ли массив динамическим
<a href="#">ArrayMaximum</a>	Поиск элемента с максимальным значением
<a href="#">ArrayMinimum</a>	Поиск элемента с минимальным значением
<a href="#">ArrayPrint</a>	Выводит в журнал массив простого типа или простой структуры
<a href="#">ArrayRange</a>	Возвращает число элементов в указанном измерении массива
<a href="#">ArrayResize</a>	Устанавливает новый размер в первом измерении массива
<a href="#">ArrayInsert</a>	Вставляет в массив-приемник из массива-источника указанное число элементов
<a href="#">ArrayRemove</a>	Удаляет из массива указанное число элементов начиная с указанного индекса
<a href="#">ArrayReverse</a>	Разворачивает в массиве указанное число элементов начиная с указанного индекса

<a href="#"><u>ArraySetAsSeries</u></a>	Устанавливает направление индексирования в массиве
<a href="#"><u>ArraySize</u></a>	Возвращает количество элементов в массиве
<a href="#"><u>ArraySort</u></a>	Сортировка числовых массивов по первому измерению
<a href="#"><u>ArraySwap</u></a>	Обменивает между собой содержимое двух динамических массивов одного типа

## ArrayBsearch

Ищет указанное значение в [отсортированном](#) по возрастанию многомерном числовом массиве. Поиск производится по элементам первого измерения.

### Для поиска в массиве типа double

```
int ArrayBsearch(
    const double& array[], // массив для поиска
    double value // что ищем
);
```

### Для поиска в массиве типа float

```
int ArrayBsearch(
    const float& array[], // массив для поиска
    float value // что ищем
);
```

### Для поиска в массиве типа long

```
int ArrayBsearch(
    const long& array[], // массив для поиска
    long value // что ищем
);
```

### Для поиска в массиве типа int

```
int ArrayBsearch(
    const int& array[], // массив для поиска
    int value // что ищем
);
```

### Для поиска в массиве типа short

```
int ArrayBsearch(
    const short& array[], // массив для поиска
    short value // что ищем
);
```

### Для поиска в массиве типа char

```
int ArrayBsearch(
    const char& array[], // массив для поиска
    char value // что ищем
);
```

### Параметры

*array[]*

[in] Числовой массив для поиска.

*value*

[in] Значение для поиска.

### Возвращаемое значение

Возвращает индекс найденного элемента. Если искомое значение не найдено, то возвращает индекс ближайшего по значению элемента.

### Примечание

Двоичный поиск обрабатывает только сортированные массивы. Для сортировки числового массива используется функция [ArraySort\(\)](#).

### Пример:

```
#property description "Скрипт на основе данных индикатора RSI выводит в окне графика"
#property description "данные о том, как часто рынок находился в зонах перекупленности"
#property description "и перепроданности в указанном промежутке времени."
//--- показем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input int           InpMAPeriod=14;                                // Период скользящей средней
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;             // Тип цены
input double         InpOversoldValue=30.0;                         // Уровень перепроданности
input double         InpOverboughtValue=70.0;                        // Уровень перекупленности
input datetime       InpDateStart=D'2012.01.01 00:00';           // Дата начала анализа
input datetime       InpDateFinish=D'2013.01.01 00:00';          // Дата конца анализа
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    double rsi_buff[]; // массив значений индикатора
    int    size=0;      // размер массива
//--- получим хэндл индикатора RSI
    ResetLastError();
    int rsi_handle=iRSI(Symbol(),Period(),InpMAPeriod,InpAppliedPrice);
    if(rsi_handle==INVALID_HANDLE)
    {
        //--- не удалось получить хэндл индикатора
        PrintFormat("Ошибка получения хэндла индикатора. Код ошибки = %d",GetLastError());
        return;
    }
//--- находимся в цикле, пока индикатор не рассчитает все свои значения
    while(BarsCalculated(rsi_handle)==-1)
    {
        //--- выходим, если пользователь принудительно завершил работу скрипта
        if(IsStopped())
            return;
        //--- задержка, чтобы индикатор успел вычислить свои значения
        Sleep(10);
    }
}
```

```

//--- скопируем значения индикатора за определенный период
ResetLastError();
if(CopyBuffer(rsi_handle,0,InpDateStart,InpDateFinish,rsi_buff)==-1)
{
    PrintFormat("Не удалось скопировать значения индикатора. Код ошибки = %d",GetLastError());
    return;
}
//--- получим размер массива
size=ArraySize(rsi_buff);
//--- отсортируем массив
ArraySort(rsi_buff);
//--- узнаем какой процент времени рынок находился в зоне перепроданности
double ovs=(double)ArrayBsearch(rsi_buff,InpOversoldValue)*100/(double)size;
//--- узнаем какой процент времени рынок находился в зоне перекупленности
double ovb=(double)(size-ArrayBsearch(rsi_buff,InpOverboughtValue))*100/(double)size;
//--- сформируем строки для вывода данных
string str="C "+TimeToString(InpDateStart,TIME_DATE)+" по "+
            +TimeToString(InpDateFinish,TIME_DATE)+" рынок находился:";
string str_ovb="в зоне перекупленности "+DoubleToString(ovb,2)+"% времени";
string str_ovs="в зоне перепроданности "+DoubleToString(ovs,2)+"% времени";
//--- отобразим данные на графике
CreateLabel("top",5,60,str,clrDodgerBlue);
CreateLabel("overbought",5,35,str_ovb,clrDodgerBlue);
CreateLabel("oversold",5,10,str_ovs,clrDodgerBlue);
//--- перерисуем график
ChartRedraw(0);
//--- задержка
Sleep(10000);
}
//+-----+
//| Вывод комментария в левый нижний угол графика |+
//+-----+
void CreateLabel(const string name,const int x,const int y,
                const string str,const color clr)
{
//--- создание метки
ObjectCreate(0,name,OBJ_LABEL,0,0,0);
//--- привязка метки к левому нижнему углу
ObjectSetInteger(0,name,OBJPROP_CORNER,CORNER_LEFT_LOWER);
//--- изменим положение точки привязки
ObjectSetInteger(0,name,OBJPROP_ANCHOR,ANCHOR_LEFT_LOWER);
//--- расстояние по оси X от точки привязки
ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x);
//--- расстояние по оси Y от точки привязки
ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y);
//--- текст метки
ObjectSetString(0,name,OBJPROP_TEXT,str);
//--- цвет текста
ObjectSetInteger(0,name,OBJPROP_COLOR,clr);
}

```

```
//--- размер текста  
ObjectSetInteger(0,name,OBJPROP_FONTSIZE,12);  
}
```

## ArrayCopy

Производит копирование одного массива в другой.

```
int ArrayCopy(
    void&      dst_array[],           // куда копируем
    const void& src_array[],          // откуда копируем
    int         dst_start=0,           // с какого индекса пишем в приемник
    int         src_start=0,           // с какого индекса копируем из источника
    int         count=WHOLE_ARRAY     // сколько элементов
);
```

### Параметры

*dst\_array[]*

[out] Массив-приемник.

*src\_array[]*

[in] Массив-источник.

*dst\_start=0*

[in] Начальный индекс для приемного массива. По умолчанию, стартовый индекс - 0.

*src\_start=0*

[in] Начальный индекс для исходного массива. По умолчанию, стартовый индекс - 0.

*count=WHOLE\_ARRAY*

[in] Количество элементов, которые нужно скопировать. По умолчанию копируется весь массив (*count=WHOLE\_ARRAY*).

### Возвращаемое значение

Возвращает количество скопированных элементов.

### Примечание

Если *count<0* либо *count>src\_size-src\_start*, то копируется весь остаток массива. Массивы копируются слева направо. Для серийных массивов правильно переопределяется стартовая позиция с учетом копирования слева направо.

Если массивы разных типов, то при копировании производится попытка преобразования каждого элемента исходного массива к типу приемного массива. Строковый массив можно скопировать только в строковый массив. Массивы [классов и структур](#), содержащих объекты, требующие инициализации, не копируются. Массив структур можно скопировать только в массив того же самого типа.

Для динамических массивов с индексацией как в [таймсерииях](#) производится автоматическое увеличение размера массива-приемника до количества копируемых данных (в случае, если количество копируемых данных превышает его размер). Автоматическое уменьшение размера массива-приемника не производится.

### Пример:

```
#property description "Индикатор выделяет цветом свечи, которые являются локальными"
```

```

#property description "максимумами и минимумами. Длину интервала для нахождения"
#property description "экстремумов можно задать при помощи входного параметра."
//--- настройки индикатора
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots 1
//---- plot
#property indicator_label1 "Extremums"
#property indicator_type1 DRAW_COLOR_CANDLES
#property indicator_color1 clrLightSteelBlue,clrRed,clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- предопределенная константа
#define INDICATOR_EMPTY_VALUE 0.0
//--- входные параметры
input int InpNum=4; // Длина полуинтервала
//--- индикаторные буфера
double ExtOpen[];
double ExtHigh[];
double ExtLow[];
double ExtClose[];
double ExtColor[];
//--- глобальные переменные
int ExtStart=0; // индекс первой свечи, которая не является экстремумом
int ExtCount=0; // количество свечей не экстремумов в данном интервале
//+-----+
//| Закрашивание свечей не экстремумов |
//+-----+
void FillCandles(const double &open[],const double &high[],
                 const double &low[],const double &close[])
{
//--- закрашиваем свечи
    ArrayCopy(ExtOpen,open,ExtStart,ExtStart,ExtCount);
    ArrayCopy(ExtHigh,high,ExtStart,ExtStart,ExtCount);
    ArrayCopy(ExtLow,low,ExtStart,ExtStart,ExtCount);
    ArrayCopy(ExtClose,close,ExtStart,ExtStart,ExtCount);
}
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,ExtOpen);
    SetIndexBuffer(1,ExtHigh);
    SetIndexBuffer(2,ExtLow);
    SetIndexBuffer(3,ExtClose);
    SetIndexBuffer(4,ExtColor,INDICATOR_COLOR_INDEX);
//--- зададим значение, которое не будет отображаться

```

```

    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VALUE);
//--- зададим имена индикаторных буферов для отображения в окне данных
    PlotIndexSetString(0,PLOT_LABEL,"Open;High;Low;Close");
//---

    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- установим прямое направление индексации в таймсериах
    ArraySetAsSeries(open,false);
    ArraySetAsSeries(high,false);
    ArraySetAsSeries(low,false);
    ArraySetAsSeries(close,false);

//--- переменная начала для расчета баров
    int start=prev_calculated;
//--- для первых InpNum*2 баров расчет не проводим
    if(start==0)
    {
        start+=InpNum*2;
        ExtStart=0;
        ExtCount=0;
    }
//--- если только что сформировался бар, то проверим следующий потенциальный экстремум
    if(rates_total-start==1)
        start--;
//--- индекс бара, который будем проверять на экстремум
    int ext;
//--- цикл расчета значений индикатора
    for(int i=start;i<rates_total-1;i++)
    {
//--- изначально на i-ом баре без отрисовки
        ExtOpen[i]=0;
        ExtHigh[i]=0;
        ExtLow[i]=0;
        ExtClose[i]=0;
//--- индекс экстремума для проверки
        ext=i-InpNum;

```

```

//--- проверка на локальный максимум
if(IsMax(high,ext))
{
    //--- пометим свечу экстремум
    ExtOpen[ext]=open[ext];
    ExtHigh[ext]=high[ext];
    ExtLow[ext]=low[ext];
    ExtClose[ext]=close[ext];
    ExtColor[ext]=1;
    //--- остальные свечи до экстремума пометим нейтральным цветом
    FillCandles(open,high,low,close);
    //--- изменяем значения переменных
    ExtStart=ext+1;
    ExtCount=0;
    //--- переходим к следующей итерации
    continue;
}
//--- проверка на локальный минимум
if(IsMin(low,ext))
{
    //--- пометим свечу экстремум
    ExtOpen[ext]=open[ext];
    ExtHigh[ext]=high[ext];
    ExtLow[ext]=low[ext];
    ExtClose[ext]=close[ext];
    ExtColor[ext]=2;
    //--- остальные свечи до экстремума пометим нейтральным цветом
    FillCandles(open,high,low,close);
    //--- изменяем значения переменных
    ExtStart=ext+1;
    ExtCount=0;
    //--- переходим к следующей итерации
    continue;
}
//--- увеличиваем количество не экстремумов в данном интервале
ExtCount++;
}

//--- return value of prev_calculated for next call
return(rates_total);
}

//-----+
//| Является ли текущий элемент массива локальным максимумом |
//-----+
bool IsMax(const double &price[],const int ind)
{
    //--- переменная начала интервала
    int i=ind-InpNum;
    //--- переменная окончания интервала
    int finish=ind+InpNum+1;
}

```

```
//--- проверка для первой половины интервала
for(;i<ind;i++)
{
    if(price[ind]<=price[i])
        return (false);
}

//--- проверка для второй половины интервала
for(i=ind+1;i<finish;i++)
{
    if(price[ind]<=price[i])
        return (false);
}

//--- это экстремум
return (true);
}

//+-----+
//| Является ли текущий элемент массива локальным минимумом |
//+-----+
bool IsMin(const double &price[],const int ind)
{
//--- переменная начала интервала
int i=ind-InpNum;
//--- переменная окончания интервала
int finish=ind+InpNum+1;
//--- проверка для первой половины интервала
for(;i<ind;i++)
{
    if(price[ind]>=price[i])
        return (false);
}

//--- проверка для второй половины интервала
for(i=ind+1;i<finish;i++)
{
    if(price[ind]>=price[i])
        return (false);
}

//--- это экстремум
return (true);
}
```

## ArrayCompare

Возвращает результат сравнения двух массивов одинакового типа. Может использоваться для сравнения массивов [простых типов](#) или пользовательских структур, не имеющих [сложных объектов](#) - то есть не содержащих [строк](#), [динамических массивов](#), классов или других структур содержащих сложные объекты.

```
int ArrayCompare(
    const void& array1[],           // первый массив
    const void& array2[],           // второй массив
    int         start1=0,           // начальное смещение в первом массиве
    int         start2=0,           // начальное смещение во втором массиве
    int         count=WHOLE_ARRAY  // количество элементов для сравнения
);
```

### Параметры

*array1[]*

[in] Первый массив.

*array2[]*

[in] Второй массив.

*start1=0*

[in] Начальный индекс элемента в первом массиве, с которого начнется сравнение. По умолчанию стартовый индекс - 0.

*start2=0*

[in] Начальный индекс элемента во втором массиве, с которого начнется сравнение. По умолчанию стартовый индекс - 0.

*count=WHOLE\_ARRAY*

[in] Количество элементов, которые нужно сравнить. По умолчанию в сравнении участвуют все элементы обоих массивов (*count=WHOLE\_ARRAY*).

### Возвращаемое значение

- -1, если *array1[]* меньше *array2[]*
- 0, если *array1[]* и *array2[]* равны
- 1, если *array1[]* больше *array2[]*
- -2, при возникновении ошибки из-за несовместимости типов сравниваемых массивов, или при значениях *start1*, *start2* или *count*, приводящих к выходу за пределы массива.

### Примечание

При разных размерах сравниваемых массивов и с указанным значением *count=WHOLE\_ARRAY* для случая, когда один массив является точным подмножеством другого, функция не вернёт 0 (массивы не будут считаться равными). В этом случае будет возвращен результат сравнения размеров этих массивов: -1, если размер *array1[]* меньше размера *array2[]* или 1 в противном случае.

## ArrayFree

Освобождает буфер любого динамического массива и устанавливает размер нулевого измерения в 0.

```
void ArrayFree(
    void& array[]          // массив
);
```

### Параметры

*array[]*  
 [in] Динамический массив.

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

При написании скриптов и индикаторов необходимость в использовании функции `ArrayFree()` может возникнуть не часто: так как при завершении работы скрипта вся использованная память сразу же освобождается, а в пользовательских индикаторах основная работа с массивами представляет собою доступ к индикаторным буферам, размеры которых автоматически управляются исполняющей подсистемой терминала.

Если в программе необходимо самостоятельно управлять памятью в сложных динамических условиях, то функция `ArrayFree()` позволит явным образом и немедленно освобождать память, занятую ненужным уже динамическим массивом.

### Пример:

```
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
#include <Controls\Label.mqh>
#include <Controls\ComboBox.mqh>
//--- предопределенные константы
#define X_START 0
#define Y_START 0
#define X_SIZE 280
#define Y_SIZE 300
//+-----+
//| Класс диалога для работы с памятью |
//+-----+
class CMemoryControl : public CAppDialog
{
private:
//--- размер массива
int             m_arr_size;
//--- массивы
char            m_arr_char[];
int             m_arr_int[];
```

```

float           m_arr_float[];
double          m_arr_double[];
long            m_arr_long[];
//--- надписи
CLabel          m_lbl_memory_physical;
CLabel          m_lbl_memory_total;
CLabel          m_lbl_memory_available;
CLabel          m_lbl_memory_used;
CLabel          m_lbl_array_size;
CLabel          m_lbl_array_type;
CLabel          m_lbl_error;
CLabel          m_lbl_change_type;
CLabel          m_lbl_add_size;
//--- КНОПКИ
CButton         m_button_add;
CButton         m_button_free;
//--- СПИСКИ
CComboBox        m_combo_box_step;
CComboBox        m_combo_box_type;
//--- текущее значение типа массива из списка
int             m_combo_box_type_value;

public:
    CMemoryControl(void);
    ~CMemoryControl(void);
//--- метод создания объекта класса
virtual bool    Create(const long chart,const string name,const int subwin,const
//--- обработчик событий графика
virtual bool    OnEvent(const int id,const long &lparam,const double &dparam,const
protected:
//--- создание надписи
bool            CreateLabel(CLabel &lbl,const string name,const int x,const int y);
//--- создание элементов управления
bool            CreateButton(CButton &button,const string name,const int x,const int y);
bool            CreateComboBoxStep(void);
bool            CreateComboBoxType(void);
//--- обработчики событий
void            OnClickButtonAdd(void);
void            OnClickButtonFree(void);
void            OnChangeComboBoxType(void);
//--- методы работы с текущим массивом
void            CurrentArrayFree(void);
bool            CurrentArrayAdd(void);
};

//-----+
//| Освобождение памяти текущего массива
//-----+
void CMemoryControl::CurrentArrayFree(void)

```

```

{
//--- сброс размера массива
m_arr_size=0;
//--- освобождение массива
if(m_combo_box_type_value==0)
    ArrayFree(m_arr_char);
if(m_combo_box_type_value==1)
    ArrayFree(m_arr_int);
if(m_combo_box_type_value==2)
    ArrayFree(m_arr_float);
if(m_combo_box_type_value==3)
    ArrayFree(m_arr_double);
if(m_combo_box_type_value==4)
    ArrayFree(m_arr_long);
}

//+-----+
// | Попытка добавления памяти для текущего массива |
//+-----+
bool CMemoryControl::CurrentArrayAdd(void)
{
//--- если размер используемой памяти больше чем размер физической памяти, то выходим
if(TerminalInfoInteger(TERMINAL_MEMORY_PHYSICAL)/TerminalInfoInteger(TERMINAL_MEMORY_VIRTUAL)>1)
    return(false);

//--- попытка выделения памяти в зависимости от текущего типа
if(m_combo_box_type_value==0 && ArrayResize(m_arr_char,m_arr_size)==-1)
    return(false);
if(m_combo_box_type_value==1 && ArrayResize(m_arr_int,m_arr_size)==-1)
    return(false);
if(m_combo_box_type_value==2 && ArrayResize(m_arr_float,m_arr_size)==-1)
    return(false);
if(m_combo_box_type_value==3 && ArrayResize(m_arr_double,m_arr_size)==-1)
    return(false);
if(m_combo_box_type_value==4 && ArrayResize(m_arr_long,m_arr_size)==-1)
    return(false);

//--- память выделена
return(true);
}

//+-----+
// | Обработка событий |
//+-----+
EVENT_MAP_BEGIN(CMemoryControl)
ON_EVENT(ON_CLICK,m_button_add,OnClickButtonAdd)
ON_EVENT(ON_CLICK,m_button_free,OnClickButtonFree)
ON_EVENT(ON_CHANGE,m_combo_box_type,OnChangeComboBoxType)
EVENT_MAP_END(CAppDialog)
//+-----+
// | Конструктор |
//+-----+
CMemoryControl::CMemoryControl(void)

```

```

{
}

//+-----+
// | Деструктор
//+-----+

CMemoryControl::~CMemoryControl (void)
{
}

//+-----+
// | Метод создания объекта класса
//+-----+

bool CMemoryControl::Create(const long chart,const string name,const int subwin,
                            const int x1,const int y1,const int x2,const int y2)
{
    //--- создание объекта базового класса
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);

    //--- подготовка строк для надписей
    string str_physical="Memory physical = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_PHYSICAL);
    string str_total="Memory total = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_TOTAL);
    string str_available="Memory available = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_AVAILABLE);
    string str_used="Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_USED);
    //--- создание надписей
    if(!CreateLabel(m_lbl_memory_physical,"physical_label",X_START+10,Y_START+5,str_physical))
        return(false);
    if(!CreateLabel(m_lbl_memory_total,"total_label",X_START+10,Y_START+30,str_total,12,clrBlack))
        return(false);
    if(!CreateLabel(m_lbl_memory_available,"available_label",X_START+10,Y_START+55,str_available))
        return(false);
    if(!CreateLabel(m_lbl_memory_used,"used_label",X_START+10,Y_START+80,str_used,12,clrBlack))
        return(false);
    if(!CreateLabel(m_lbl_array_type,"type_label",X_START+10,Y_START+105,"Array type ="))
        return(false);
    if(!CreateLabel(m_lbl_array_size,"size_label",X_START+10,Y_START+130,"Array size ="))
        return(false);
    if(!CreateLabel(m_lbl_error,"error_label",X_START+10,Y_START+155,"",12,clrRed))
        return(false);
    if(!CreateLabel(m_lbl_change_type,"change_type_label",X_START+10,Y_START+185,"Change type"))
        return(false);
    if(!CreateLabel(m_lbl_add_size,"add_size_label",X_START+10,Y_START+210,"Add to array"))
        return(false);
    //--- создание элементов управления
    if(!CreateButton(m_button_add,"add_button",X_START+15,Y_START+245,"Add",12,clrBlue))
        return(false);
    if(!CreateButton(m_button_free,"free_button",X_START+75,Y_START+245,"Free",12,clrBlue))
        return(false);
    if(!CreateComboBoxType())
        return(false);
    if(!CreateComboBoxStep())
        return(false);
}

```

```

        return(false);
//--- инициализация переменной
m_arr_size=0;
//--- успешное выполнение
return(true);
}

//+-----+
//| Создание кнопки
//+-----+
bool CMemoryControl::CreateButton(CButton &button,const string name,const int x,
                                    const int y,const string str,const int font_size,
                                    const int clr)
{
//--- создание кнопки
if(!button.Create(m_chart_id,name,m_subwin,x,y,x+50,y+20))
    return(false);
//--- текст
if(!button.Text(str))
    return(false);
//--- размер шрифта
if(!button.FontSize(font_size))
    return(false);
//--- цвет надписи
if(!button.Color(clr))
    return(false);
//--- добавляем кнопку в элементы контроля
if(!Add(button))
    return(false);
//--- успешное выполнение
return(true);
}

//+-----+
//| Создание списка для размера массива
//+-----+
bool CMemoryControl::CreateComboBoxStep(void)
{
//--- создание списка
if(!m_combo_box_step.Create(m_chart_id,"step_combobox",m_subwin,X_START+100,Y_START+100))
    return(false);
//--- добавление элементов в список
if(!m_combo_box_step.ItemAdd("100 000",100000))
    return(false);
if(!m_combo_box_step.ItemAdd("1 000 000",1000000))
    return(false);
if(!m_combo_box_step.ItemAdd("10 000 000",10000000))
    return(false);
if(!m_combo_box_step.ItemAdd("100 000 000",100000000))
    return(false);
//--- установим текущий элемент списка
}

```

```

if(!m_combo_box_step.SelectByValue(1000000))
    return(false);
//--- добавляем список в элементы контроля
if(!Add(m_combo_box_step))
    return(false);
//--- успешное выполнение
return(true);
}

//+-----+
//| Создание списка для типа массива |
//+-----+
bool CMemoryControl::CreateComboBoxType(void)
{
//--- создание списка
if(!m_combo_box_type.Create(m_chart_id,"type_combobox",m_subwin,X_START+100,Y_START+100,100,50))
    return(false);
//--- добавление элементов в список
if(!m_combo_box_type.ItemAdd("char",0))
    return(false);
if(!m_combo_box_type.ItemAdd("int",1))
    return(false);
if(!m_combo_box_type.ItemAdd("float",2))
    return(false);
if(!m_combo_box_type.ItemAdd("double",3))
    return(false);
if(!m_combo_box_type.ItemAdd("long",4))
    return(false);
//--- установим текущий элемент списка
if(!m_combo_box_type.SelectByValue(3))
    return(false);
//--- запомним текущий элемент списка
m_combo_box_type_value=3;
//--- добавляем список в элементы контроля
if(!Add(m_combo_box_type))
    return(false);
//--- успешное выполнение
return(true);
}

//+-----+
//| Создание надписи |
//+-----+
bool CMemoryControl::CreateLabel(CLabel &lbl,const string name,const int x,
                                const int y,const string str,const int font_size,
                                const int clr)
{
//--- создание надписи
if(!lbl.Create(m_chart_id,name,m_subwin,x,y,0,0))
    return(false);
//--- текст
}

```

```

if(!lbl.Text(str))
    return(false);
//--- размер шрифта
if(!lbl.FontSize(font_size))
    return(false);
//--- цвет
if(!lbl.Color(clr))
    return(false);
//--- добавляем надпись в элементы контроля
if(!Add(lbl))
    return(false);
//--- succeed
return(true);
}

//+-----+
//| Обработчик события нажатия на кнопку "Add" |
//+-----+
void CMemoryControl::OnClickButtonAdd(void)
{
//--- увеличим размер массива
m_arr_size+=(int)m_combo_box_step.Value();
//--- пытаемся выделить память под текущий массив
if(CurrentArrayAdd())
{
//--- память выделена, выводим текущее состояние на экран
m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_AVAILABLE));
m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_USED));
m_lbl_array_size.Text("Array size = "+IntegerToString(m_arr_size));
m_lbl_error.Text("");
}
else
{
//--- не удалось выделить память, выводим сообщение об ошибке
m_lbl_error.Text("Array is too large, error!");
//--- вернем предыдущий размер массива
m_arr_size-=(int)m_combo_box_step.Value();
}
}

//+-----+
//| Обработчик события нажатия на кнопку "Free" |
//+-----+
void CMemoryControl::OnClickButtonFree(void)
{
//--- освобождаем память текущего массива
CurrentArrayFree();
//--- выведем текущее состояние на экран
m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_AVAILABLE));
m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_USED));
m_lbl_array_size.Text("Array size = 0");
}

```

```

        m_lbl_error.Text("");
    }

//+-----+
//| Обработчик события изменения списка |
//+-----+

void CMemoryControl::OnChangeComboBoxType(void)
{
    //--- проверка, изменился ли тип массива
    if(m_combo_box_type.Value()!=m_combo_box_type_value)
    {
        //--- освобождаем память текущего массива
        OnClickButtonFree();

        //--- работаем с другим типом массива
        m_combo_box_type_value=(int)m_combo_box_type.Value();
        //--- выведем на экран новый тип массива
        if(m_combo_box_type_value==0)
            m_lbl_array_type.Text("Array type = char");
        if(m_combo_box_type_value==1)
            m_lbl_array_type.Text("Array type = int");
        if(m_combo_box_type_value==2)
            m_lbl_array_type.Text("Array type = float");
        if(m_combo_box_type_value==3)
            m_lbl_array_type.Text("Array type = double");
        if(m_combo_box_type_value==4)
            m_lbl_array_type.Text("Array type = long");
    }
}

//--- объект класса CMemoryControl
CMemoryControl ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+

int OnInit()
{
    //--- создание диалога
    if(!ExtDialog.Create(0,"MemoryControl",0,X_START,Y_START,X_SIZE,Y_SIZE))
        return(INIT_FAILED);
    //--- запуск
    ExtDialog.Run();
    //---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function |
//+-----+

void OnDeinit(const int reason)
{
    //---
    ExtDialog.Destroy(reason);
}

```

```
    }

//+-----+
//| Expert chart event function
//+-----+

void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## ArrayGetAsSeries

Проверяет направление индексации массива.

```
bool ArrayGetAsSeries(
    const void& array[] // массив для проверки
);
```

### Параметры

*array[]*  
[in] Проверяемый массив.

### Возвращаемое значение

Возвращает [true](#), если у указанного массива установлен флаг AS\_SERIES, то есть доступ к массиву осуществляется задом наперед как в таймсерии. [Таймсерия](#) отличается от обычного массива тем, что индексация элементов таймсерии производится от конца массива к началу (от самых свежих данных к самым старым).

### Примечание

Для проверки массива на принадлежность к таймсерии следует применять функцию [ArrayIsSeries\(\)](#). Массивы ценовых данных, переданных в качестве входных параметров в функцию [OnCalculate\(\)](#), не обязательно имеют направление индексации как у таймсерий. Нужное направление индексации можно установить функцией [ArraySetAsSeries\(\)](#).

### Пример:

```
#property description "Индикатор вычисляет абсолютные значения разницы между ценами"
#property description "Open и Close или High и Low, и отображает их в отдельном подокне"
#property description "в виде гистограммы."
//--- настройки индикатора
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot
#property indicator_type1 DRAW_HISTOGRAM
#property indicator_style1 STYLE_SOLID
#property indicator_width1 3
//--- входные параметры
input bool InpAsSeries=true; // Направление индексации в индикаторном буфере
input bool InpPrices=true; // Цены для расчета (true - Open,Close; false - High,Low)
//--- индикаторный буфер
double ExtBuffer[];
//+-----+
//| Вычисление значений индикатора |
//+-----+
void CandleSizeOnBuffer(const int rates_total,const int prev_calculated,
                       const double &first[],const double &second[],double &buffer[])
{
//--- переменная начала для расчета баров
```

```

int start=prev_calculated;
//--- если значения индикатора уже были рассчитаны на предыдущем тике, то работаем на
if(prev_calculated>0)
    start--;
//--- определим направление индексации в массивах
bool as_series_first=ArrayGetAsSeries(first);
bool as_series_second=ArrayGetAsSeries(second);
bool as_series_buffer=ArrayGetAsSeries(buffer);
//--- изменим направление индексации на прямое, если необходимо
if(as_series_first)
    ArraySetAsSeries(first,false);
if(as_series_second)
    ArraySetAsSeries(second,false);
if(as_series_buffer)
    ArraySetAsSeries(buffer,false);
//--- рассчитаем значения индикатора
for(int i=start;i<rates_total;i++)
    buffer[i]=MathAbs(first[i]-second[i]);
}
//----------------------------------------------------------------------------------------------------------------+
//| Custom indicator initialization function
//----------------------------------------------------------------------------------------------------------------+
int OnInit()
{
//--- привязка индикаторных буферов
SetIndexBuffer(0,ExtBuffer);
//--- установим направление индексации в индикаторном буфере
ArraySetAsSeries(ExtBuffer,InpAsSeries);
//--- проверяем для каких цен рассчитывается индикатор
if(InpPrices)
{
//--- цены Open и Close
PlotIndexSetString(0,PLOT_LABEL,"BodySize");
//--- установим цвет индикатора
PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrOrange);
}
else
{
//--- цены High и Low
PlotIndexSetString(0,PLOT_LABEL,"ShadowSize");
//--- установим цвет индикатора
PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrDodgerBlue);
}
//---
return(INIT_SUCCEEDED);
}
//----------------------------------------------------------------------------------------------------------------+
//| Custom indicator iteration function
//----------------------------------------------------------------------------------------------------------------+

```

```
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- расчет индикатора в зависимости от значения флага
    if(IInpPrices)
        CandleSizeOnBuffer(rates_total,prev_calculated,open,close,ExtBuffer);
    else
        CandleSizeOnBuffer(rates_total,prev_calculated,high,low,ExtBuffer);
    //--- return value of prev_calculated for next call
    return(rates_total);
}
```

#### Смотри также

[Доступ к таймсериям](#), [ArraySetAsSeries](#)

## ArrayInitialize

Инициализирует числовой массив указанным значением.

**Для инициализации массива типа char**

```
int ArrayInitialize(
    char array[],      // инициализируемый массив
    char value         // значение, которое будет установлено
);
```

**Для инициализации массива типа short**

```
int ArrayInitialize(
    short array[],     // инициализируемый массив
    short value        // значение, которое будет установлено
);
```

**Для инициализации массива типа int**

```
int ArrayInitialize(
    int array[],        // инициализируемый массив
    int value           // значение, которое будет установлено
);
```

**Для инициализации массива типа long**

```
int ArrayInitialize(
    long array[],       // инициализируемый массив
    long value          // значение, которое будет установлено
);
```

**Для инициализации массива типа float**

```
int ArrayInitialize(
    float array[],      // инициализируемый массив
    float value         // значение, которое будет установлено
);
```

**Для инициализации массива типа double**

```
int ArrayInitialize(
    double array[],      // инициализируемый массив
    double value         // значение, которое будет установлено
);
```

**Для инициализации массива типа bool**

```
int ArrayInitialize(
    bool array[],        // инициализируемый массив
    bool value           // значение, которое будет установлено
);
```

### Для инициализации массива типа uint

```
int ArrayInitialize(
    uint array[],      // инициализируемый массив
    uint value         // значение, которое будет установлено
);
```

#### Параметры

*array[]*

[out] Числовой массив, который нужно инициализировать.

*value*

[in] Новое значение, которое нужно установить всем элементам массива.

#### Возвращаемое значение

Количество инициализированных элементов.

#### Примечание

Функция [ArrayResize\(\)](#) позволяет задать для массива размер с некоторым запасом для его будущего увеличения без физического перераспределения памяти. Это сделано для улучшения быстродействия, так как операции по распределению памяти являются достаточно медленными.

Инициализация массива выражением `ArrayInitialize(array, init_val)` не означает инициализацию этим же значением и элементов резерва, выделенного для этого массива. При последующих увеличениях размера массива *array* функцией `ArrayResize()` в пределах текущего резерва, в конец массива добавляются элементы, значения которых не определены и, чаще всего, не равны *init\_val*.

#### Пример:

```
void OnStart()
{
//--- динамический массив
double array[];
//--- зададим размер массива для 100 элементов и зарезервируем еще буфер в 10 элементов
ArrayResize(array,100,10);
//--- инициализируем элементы массива значением EMPTY_VALUE=DBL_MAX
ArrayInitialize(array,EMPTY_VALUE);
Print("Значения последних 10 элементов массива после инициализации");
for(int i=90;i<100;i++)
printf("array[%d] = %G",i,array[i]);
//--- увеличим размер массива на 5 элементов
ArrayResize(array,105);
Print("Значения последних 10 элементов массива после ArrayResize(array,105)");
//--- значения последних 5 элементов были получены из буфера резерва
for(int i=95;i<105;i++)
printf("array[%d] = %G",i,array[i]);
}
```

## ArrayFill

Заполняет числовой массив указанным значением.

```
void ArrayFill(
    void& array[],           // массив
    int start,               // индекс начального элемента
    int count,               // количество элементов
    void value                // значение, которым заполняется массив
);
```

### Параметры

*array[]*

[out] Массив простого типа ([char](#), [uchar](#), [short](#), [ushort](#), [int](#), [uint](#), [long](#), [ulong](#), [bool](#), [color](#), [datetime](#), [float](#), [double](#)).

*start*

[in] Индекс начального элемента (с какого элемента заполнять). При этом не учитывается установленный [флаг серииности](#).

*count*

[in] Количество элементов, которое следует заполнить.

*value*

[in] Значение, которым заполняется массив.

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

При вызове функции `ArrayFill()` всегда подразумевается обычное направление индексации - слева направо, то есть изменение порядка доступа к элементам массива с помощью функции [ArraySetAsSeries\(\)](#) не принимается во внимание.

Многомерный массив при обработке функцией `ArrayFill()` представляется одномерным, например, массив `array[2][4]` обрабатывается как `array[8]`, поэтому при работе с этим массивом допустимо указать индекс начального элемента равным 5. Таким образом, вызов `ArrayFill(array, 5, 2, 3.14)` для массива `array[2][4]` заполнит значением 3.14 элементы массива `array[1][1]` и `array[1][2]`.

### Пример:

```
void OnStart()
{
//--- объявляем динамический массив
int a[];
//--- устанавливаем размер
ArrayResize(a,10);
//--- заполняем начальные 5 элементов значением 123
ArrayFill(a,0,5,123);
```

```
//--- заполняем 5 элементов (начиная с 5-го) значением 456
ArrayFill(a,5,5,456);
//--- выводим значения всех элементов
for(int i=0;i<ArraySize(a);i++)
    printf("a[%d] = %d",i,a[i]);
}
```

## ArrayIsDynamic

Проверяет, является ли массив динамическим.

```
bool ArrayIsDynamic(
    const void& array[] // проверяемый массив
);
```

### Параметры

*array[]*  
[in] Проверяемый массив.

### Возвращаемое значение

Возвращает true, если указанный массив является [динамическим](#), иначе возвращается false.

### Пример:

```
#property description "Этот индикатор не вычисляет значений, а один раз пытается применить их к трем массивам: динамическому, статическому и индикаторному буферу. Результаты выводятся в журнал экспертов."
//--- настройки индикатора
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- глобальные переменные
double ExtDynamic[]; // динамический массив
double ExtStatic[100]; // статический массив
bool ExtFlag=true; // флаг
double ExtBuff[]; // индикаторный буфер
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- выделим память для массива
    ArrayResize(ExtDynamic,100);
//--- indicator buffers mapping
    SetIndexBuffer(0,ExtBuff);
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double &price[])
{}
```

```
{  
//--- проведем анализ один раз  
if(ExtFlag)  
{  
//--- попробуем освободить память для массивов  
//--- 1. Динамический массив  
Print("=====+");  
Print("1. Проверка динамического массива:");  
Print("Размер до освобождения памяти = ",ArraySize(ExtDynamic));  
Print("Это динамический массив = ",ArrayIsDynamic(ExtDynamic) ? "Да" : "Нет");  
//--- пытаемся освободить память массива  
ArrayFree(ExtDynamic);  
Print("Размер после освобождения памяти = ",ArraySize(ExtDynamic));  
//--- 2. Статический массив  
Print("2. Проверка статического массива:");  
Print("Размер до освобождения памяти = ",ArraySize(ExtStatic));  
Print("Это динамический массив = ",ArrayIsDynamic(ExtStatic) ? "Да" : "Нет");  
//--- пытаемся освободить память массива  
ArrayFree(ExtStatic);  
Print("Размер после освобождения памяти = ",ArraySize(ExtStatic));  
//--- 3. Индикаторный буфер  
Print("3. Проверка индикаторного буфера:");  
Print("Размер до освобождения памяти = ",ArraySize(ExtBuff));  
Print("Это динамический массив = ",ArrayIsDynamic(ExtBuff) ? "Да" : "Нет");  
//--- пытаемся освободить память массива  
ArrayFree(ExtBuff);  
Print("Размер после освобождения памяти = ",ArraySize(ExtBuff));  
//--- изменим значение флага  
ExtFlag=false;  
}  
//--- return value of prev_calculated for next call  
return(rates_total);  
}
```

## Смотри также

[Доступ к таймсериям и индикаторам](#)

## ArrayIsSeries

Проверяет, является ли массив таймсерий.

```
bool ArrayIsSeries(
    const void& array[] // проверяемый массив
);
```

### Параметры

*array[]*  
[in] Проверяемый массив.

### Возвращаемое значение

Возвращается `true`, если проверяемый массив является массивом-таймсерий, иначе возвращается `false`. Массивы, передаваемые в качестве параметра функции [OnCalculate\(\)](#), необходимо проверять на порядок доступа к элементам массива функцией [ArrayGetAsSeries\(\)](#).

### Пример:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- plot Label1
#property indicator_label1 "Label1"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- indicator buffers
double Label1Buffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
//---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
```

```
const long &tick_volume[],
const long &volume[],
const int &spread[])
{
//---

if(ArrayIsSeries(open))
    Print("open[] is timeseries");
else
    Print("open[] is not timeseries!!!");

//--- return value of prev_calculated for next call
return(rates_total);
}
```

Смотри также

[Доступ к таймсериям и индикаторам](#)

## ArrayMaximum

Ищет максимальный элемент в первом измерении многомерного числового массива.

```
int ArrayMaximum(
    const void& array[],           // массив для поиска
    int start=0,                  // с какого индекса начинаем поиск
    int count=WHOLE_ARRAY         // количество проверяемых
);
```

### Параметры

*array[]*

[in] Числовой массив, в котором производится поиск.

*start=0*

[in] Начальный индекс для поиска.

*count=WHOLE\_ARRAY*

[in] Количество элементов для поиска. По умолчанию, ищет во всем массиве (count=WHOLE\_ARRAY).

### Возвращаемое значение

Возвращает индекс найденного элемента без учета серийности массива. В случае неудачи функция возвращает -1.

### Примечание

Поиск максимального элемента производится с учетом значения флага AS\_SERIES.

Функции ArrayMaximum и ArrayMinimum принимают в качестве параметра массив любой размерности, при этом поиск происходит только по первому (нулевому) измерению.

### Пример:

```
#property description "Индикатор отображает свечи со старшего таймфрейма на текущем."
//--- настройки индикатора
#property indicator_chart_window
#property indicator_buffers 16
#property indicator_plots 8
//--- plot 1
#property indicator_label1 "BearBody"
#property indicator_color1 clrSeaGreen,clrSeaGreen
//--- plot 2
#property indicator_label2 "BearBodyEnd"
#property indicator_color2 clrSeaGreen,clrSeaGreen
//--- plot 3
#property indicator_label3 "BearShadow"
#property indicator_color3 clrSalmon,clrSalmon
//--- plot 4
#property indicator_label4 "BearShadowEnd"
#property indicator_color4 clrSalmon,clrSalmon
```

```

//---- plot 5
#property indicator_label5 "BullBody"
#property indicator_color5 clrOlive,clrOlive
//---- plot 6
#property indicator_label6 "BullBodyEnd"
#property indicator_color6 clrOlive,clrOlive
//---- plot 7
#property indicator_label7 "BullShadow"
#property indicator_color7 clrSkyBlue,clrSkyBlue
//---- plot 8
#property indicator_label8 "BullShadowEnd"
#property indicator_color8 clrSkyBlue,clrSkyBlue
//--- предопределенная константа
#define INDICATOR_EMPTY_VALUE 0.0
//--- входные параметры
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H4; // Таймфрейм для расчета индикатора
input datetime InpDateStart=D'2013.01.01 00:00'; // Дата начала анализа
//--- индикаторные буферы для медвежьих свечей
double ExtBearBodyFirst[];
double ExtBearBodySecond[];
double ExtBearBodyEndFirst[];
double ExtBearBodyEndSecond[];
double ExtBearShadowFirst[];
double ExtBearShadowSecond[];
double ExtBearShadowEndFirst[];
double ExtBearShadowEndSecond[];
//--- индикаторные буферы для бычьих свечей
double ExtBullBodyFirst[];
double ExtBullBodySecond[];
double ExtBullBodyEndFirst[];
double ExtBullBodyEndSecond[];
double ExtBullShadowFirst[];
double ExtBullShadowSecond[];
double ExtBullShadowEndFirst[];
double ExtBullShadowEndSecond[];
//--- глобальные переменные
datetime ExtTimeBuff[]; // буфер для времени с высшего таймфрейма
int ExtSize=0; // размер буфера времени
int ExtCount=0; // индекс внутри буфера времени
int ExtStartPos=0; // начальная позиция для расчета индикатора
bool ExtStartFlag=true; // вспомогательный флаг для получения начальной позиции
datetime ExtCurrentTime[1]; // последнее время формирования бара со старшего таймфрейма
datetime ExtLastTime; // последнее время со старшего таймфрейма, для которого индикатор
bool ExtBearFlag=true; // флаг для определения порядка записи данных в медвежьи индикаторы
bool ExtBullFlag=true; // флаг для определения порядка записи данных в бычьи индикаторы
int ExtIndexMax=0; // индекс максимального элемента в массиве
int ExtIndexMin=0; // индекс минимального элемента в массиве
int ExtDirectionFlag=0; // направление движения цены у текущей свечи
//--- отступ между ценой открытия и закрытия свечи для правильной отрисовки

```

```

const double ExtEmptyBodySize=0.2*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+-----+
//| Закрашивание основной части свечи |
//+-----+
void FillCandleMain(const double &open[],const double &close[],
                     const double &high[],const double &low[],
                     const int start,const int last,const int fill_index,
                     int &index_max,int &index_min)
{
//--- найдем индексы максимального и минимального элементов в массивах
    index_max=ArrayMaximum(high,ExtStartPos,last-start+1); // максимум в High
    index_min=ArrayMinimum(low,ExtStartPos,last-start+1); // минимум в Low
//--- определим сколько баров с текущего таймфрейма будем закрашивать
    int count=fill_index-start+1;
//--- если цена закрытия на первом баре больше цены закрытия на последнем - свеча медвежья
    if(open[start]>close[last])
    {
//--- если до этого свеча была бычьей, то очищаем значения бычьих индикаторных буферов
        if(ExtDirectionFlag!=-1)
            ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond);
//--- медвежья свеча
        ExtDirectionFlag=-1;
//--- формируем свечку
        FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,
                       close[last],high[index_max],low[index_min],start,count,ExtBearFlag);
//--- выход из функции
        return;
    }
//--- если цена закрытия на первом баре меньше цены закрытия на последнем - свеча бычья
    if(open[start]<close[last])
    {
//--- если до этого свеча была медвежьей, то очищаем значения медвежьих индикаторных буферов
        if(ExtDirectionFlag!=1)
            ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond);
//--- бычья свеча
        ExtDirectionFlag=1;
//--- формируем свечку
        FormCandleMain(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond,
                       open[start],high[index_max],low[index_min],start,count,ExtBullFlag);
//--- выход из функции
        return;
    }
//--- если попали в эту часть функции, то значит цена открытия на первом баре равняется
//--- цене закрытия на последнем баре; будем считать такую свечу медвежьей
//--- если до этого свеча была бычьей, то очищаем значения бычьих индикаторных буферов
    if(ExtDirectionFlag!=-1)
        ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond);
//--- медвежья свеча
    ExtDirectionFlag=-1;
}

```

```

//--- если цены закрытия и цены открытия равны, то используем сдвиг для корректного отображения
if(high[index_max]!=low[index_min])
    FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,
                    open[start]-ExtEmptyBodySize,high[index_max],low[index_min],start,fill_index);
else
    FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,
                    open[start],open[start]-ExtEmptyBodySize,high[index_max],
                    high[index_max]-ExtEmptyBodySize,start,count,ExtBearFlag);
}

//+-----+
//| Закрашивание конца свечи
//+-----+
void FillCandleEnd(const double &open[],const double &close[],
                    const double &high[],const double &low[],
                    const int start,const int last,const int fill_index,
                    const int index_max,const int index_min)
{
//--- если всего один бар, то не рисуем
if(last-start==0)
    return;

//--- если цена закрытия на первом баре больше цены закрытия на последнем - свеча медвежья
if(open[start]>close[last])
{
    //--- формируем конец свечи
    FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,ExtBearShadowEndSecond,
                  open[start],close[last],high[index_max],low[index_min],fill_index);
    //--- выход из функции
    return;
}

//--- если цена закрытия на первом баре меньше цены закрытия на последнем - свеча бычьего
if(open[start]<close[last])
{
    //--- формируем конец свечи
    FormCandleEnd(ExtBullBodyEndFirst,ExtBullBodyEndSecond,ExtBullShadowEndFirst,ExtBullShadowEndSecond,
                  close[last],open[start],high[index_max],low[index_min],fill_index);
    //--- выход из функции
    return;
}

//--- если попали в эту часть функции, то значит цена открытия на первом баре равняется
//--- цене закрытия на последнем баре; будем считать такую свечу медвежьей
//--- формируем конец свечи
if(high[index_max]!=low[index_min])
    FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,ExtBearShadowEndSecond,
                  open[start]-ExtEmptyBodySize,high[index_max],low[index_min],fill_index);
else
    FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,ExtBearShadowEndSecond,
                  open[start]-ExtEmptyBodySize,high[index_max],high[index_max]-ExtEmptyBodySize,fill_index);
}
//+-----+

```

```

//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- проверка периода индикатора
    if(!CheckPeriod((int)Period(),(int)InpPeriod))
        return(INIT_PARAMETERS_INCORRECT);
//--- отображение ценовых данных на переднем плане
    ChartSetInteger(0,CHART_FOREGROUND,0,1);
//--- привязка индикаторных буферов
    SetIndexBuffer(0,ExtBearBodyFirst);
    SetIndexBuffer(1,ExtBearBodySecond);
    SetIndexBuffer(2,ExtBearBodyEndFirst);
    SetIndexBuffer(3,ExtBearBodyEndSecond);
    SetIndexBuffer(4,ExtBearShadowFirst);
    SetIndexBuffer(5,ExtBearShadowSecond);
    SetIndexBuffer(6,ExtBearShadowEndFirst);
    SetIndexBuffer(7,ExtBearShadowEndSecond);
    SetIndexBuffer(8,ExtBullBodyFirst);
    SetIndexBuffer(9,ExtBullBodySecond);
    SetIndexBuffer(10,ExtBullBodyEndFirst);
    SetIndexBuffer(11,ExtBullBodyEndSecond);
    SetIndexBuffer(12,ExtBullShadowFirst);
    SetIndexBuffer(13,ExtBullShadowSecond);
    SetIndexBuffer(14,ExtBullShadowEndFirst);
    SetIndexBuffer(15,ExtBullShadowEndSecond);

//--- зададим некоторые значения свойств для построения индикатора
    for(int i=0;i<8;i++)
    {
        PlotIndexSetInteger(i,PLOT_DRAW_TYPE,DRAW_FILLING); // тип графического построения
        PlotIndexSetInteger(i,PLOT_LINE_STYLE,STYLE_SOLID); // стиль линии отрисовки
        PlotIndexSetInteger(i,PLOT_LINE_WIDTH,1);           // толщина линии отрисовки
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])

```

```

{
//--- если еще нет рассчитанных баров
if(prev_calculated==0)
{
    //--- получим время появления баров со старшего таймфрейма
    if(!GetTimeData())
        return(0);
}

//--- установим прямое направление индексации
ArraySetAsSeries(time,false);
ArraySetAsSeries(high,false);
ArraySetAsSeries(low,false);
ArraySetAsSeries(open,false);
ArraySetAsSeries(close,false);

//--- переменная начала для расчета баров
int start=prev_calculated;
//--- если бар формируется, то пересчитываем значение индикатора на нем
if(start!=0 && start==rates_total)
    start--;
//--- цикл расчета значений индикатора
for(int i=start;i<rates_total;i++)
{
    //--- заполняем i-ые элементы индикаторных буферов пустыми значениями
    FillIndicatorBuffers(i);
    //--- проводим вычисление для баров начиная с даты InpDateStart
    if(time[i]>=InpDateStart)
    {
        //--- в первый раз определим позицию с которой начнем отображать значения
        if(ExtStartFlag)
        {
            //--- запомним номер начального бара
            ExtStartPos=i;
            //--- определим первую дату со старшего таймфрейма, которая больше time[i]
            while(time[i]>=ExtTimeBuff[ExtCount])
                if(ExtCount<ExtSize-1)
                    ExtCount++;
            //--- изменим значение флага, чтобы больше не попадать в этот блок
            ExtStartFlag=false;
        }
        //--- проверка, есть ли еще в массиве элементы
        if(ExtCount<ExtSize)
        {
            //--- ждем, когда значение времени с текущего таймфрейма достигнет значени
            if(time[i]>=ExtTimeBuff[ExtCount])
            {
                //--- рисуем главную часть свечи (без закрашивания между последним и пр
                FillCandleMain(open,close,high,low,ExtStartPos,i-1,i-2,ExtIndexMax,Ext
                //--- закрашиваем конец свечи (область между последним и предпоследним
                FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,Ext
}

```

```

        //--- сдвигаем начальную позицию для рисования следующей свечи
ExtStartPos=i;
//--- увеличиваем счетчик массива
ExtCount++;
}
else
    continue;
}
else
{
//--- сбрасываем значения ошибки
ResetLastError();
//--- получаем последнюю дату со старшего таймфрейма
if(CopyTime(Symbol(),InpPeriod,0,1,ExtCurrentTime)==-1)
{
    Print("Ошибка копирования данных, код = ",GetLastError());
    return(0);
}
//--- если новая дата больше, то значит заканчиваем формирование свечки
if(ExtCurrentTime[0]>ExtLastTime)
{
    //--- очистим область между последним и предпоследним баром в главных индикаторах
    ClearEndOfBodyMain(i-1);
    //--- закрасим эту область с помощью вспомогательных индикаторных буферов
    FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIndexMin);
    //--- сдвигаем начальную позицию для рисования следующей свечи
    ExtStartPos=i;
    //--- сбросим флаг направления цены
    ExtDirectionFlag=0;
    //--- запомним новую последнюю дату
    ExtLastTime=ExtCurrentTime[0];
}
else
{
    //--- формируем свечку
    FillCandleMain(open,close,high,low,ExtStartPos,i,i,ExtIndexMax,ExtIndexMin);
}
}
}
}

//--- return value of prev_calculated for next call
return(rates_total);
}

//+-----+
//| Проверка введенного периода индикатора на корректность |
//+-----+
bool CheckPeriod(int current_period,int high_period)
{
//--- период индикатора должен быть больше таймфрейма, на котором он отображается
}

```

```

if(current_period>=high_period)
{
    Print("Ошибка! Значение периода индикатора должно быть больше значения текущего");
    return(false);
}

//--- если период индикатора - одна неделя или месяц, то период корректен
if(high_period>32768)
    return(true);

//--- приведем значения периодов к минутам
if(high_period>30)
    high_period=(high_period-16384)*60;
if(current_period>30)
    current_period=(current_period-16384)*60;

//--- период индикатора должен быть кратен таймфрейму, на котором он отображается
if(high_period%current_period!=0)
{
    Print("Ошибка! Значение периода индикатора должно быть кратным значению текущего таймфрейма");
    return(false);
}

//--- период индикатора должен превышать значение таймфрейма, на котором он отображается
if(high_period/current_period<3)
{
    Print("Ошибка! Значение периода индикатора должно превышать значение текущего таймфрейма");
    return(false);
}

//--- период индикатора корректен для текущего таймфрейма
return(true);
}

//-----+
//| Получение данных времени со старшего таймфрейма |
//-----+



bool GetTimeData(void)
{
//--- сброс значения ошибки
ResetLastError();

//--- скопируем все данные на текущее время
if(CopyTime(Symbol(), InpPeriod, InpDateStart, TimeCurrent(), ExtTimeBuff)==-1)
{
    //--- получим код ошибки
    int code=GetLastError();
    //--- распечатаем текст ошибки
    PrintFormat("Ошибка копирования данных! %s", code==4401
        ? "История еще подгружается!"
        : "Код = "+IntegerToString(code));
    //--- вернем false для повторной попытки загрузки данных
    return(false);
}

//--- получим размер массива
ExtSize=ArraySize(ExtTimeBuff);
}

```

```

//--- установим индекс цикла для массива равным нулю
ExtCount=0;
//--- установим позицию текущей свечи на данном таймфрейме равной нулю
ExtStartPos=0;
ExtStartFlag=true;
//--- запомним последнее значение времени со старшего таймфрейма
ExtLastTime=ExtTimeBuff[ExtSize-1];
//--- успешное выполнение
return(true);
}

//+-----+
//| Функция формирует основную часть свечи. В зависимости от значения |
//| флага, функция определяет, какие данные в какие массивы должны     |
//| записываться для корректного отображения.                           |
//+-----+
void FormCandleMain(double &body_fst[],double &body_snd[],
                     double &shadow_fst[],double &shadow_snd[],
                     const double fst_value,const double snd_value,
                     const double fst_extremum,const double snd_extremum,
                     const int start,const int count,const bool flag)
{
//--- проверяем значение флага
if(flag)
{
    //--- формируем тело свечи
    FormMain(body_fst,body_snd,fst_value,snd_value,start,count);
    //--- формируем тень свечи
    FormMain(shadow_fst,shadow_snd,fst_extremum,snd_extremum,start,count);
}
else
{
    //--- формируем тело свечи
    FormMain(body_fst,body_snd,snd_value,fst_value,start,count);
    //--- формируем тень свечи
    FormMain(shadow_fst,shadow_snd,snd_extremum,fst_extremum,start,count);
}
}

//+-----+
//| Функция формирует конец свечи. В зависимости от значения флага,   |
//| функция определяет, какие данные в какие массивы должны             |
//| записываться для корректного отображения.                           |
//+-----+
void FormCandleEnd(double &body_fst[],double &body_snd[],
                    double &shadow_fst[],double &shadow_snd[],
                    const double fst_value,const double snd_value,
                    const double fst_extremum,const double snd_extremum,
                    const int end,bool &flag)
{
//--- проверяем значение флага
}

```

```

if(flag)
{
    //--- формируем конец тела свечи
    FormEnd(body_fst,body_snd,fst_value,snd_value,end);
    //--- формируем конец тени свечи
    FormEnd(shadow_fst,shadow_snd,fst_extremum,snd_extremum,end);
    //--- меняем значение флага на противоположное
    flag=false;
}
else
{
    //--- формируем конец тела свечи
    FormEnd(body_fst,body_snd,snd_value,fst_value,end);
    //--- формируем конец тени свечи
    FormEnd(shadow_fst,shadow_snd,snd_extremum,fst_extremum,end);
    //--- меняем значение флага на противоположное
    flag=true;
}

//+-----+
//| Очистка конца свечи (область между последним и предпоследним
//| баром)
//+-----+
void ClearEndOfBodyMain(const int ind)
{
    ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,ind);
    ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond,ind);
}

//+-----+
//| Очистка свечи
//+-----+
void ClearCandle(double &body_fst[],double &body_snd[],double &shadow_fst[],
                 double &shadow_snd[],const int start,const int count)
{
    //--- проверка
    if(count!=0)
    {
        //--- заполняем индикаторные буфера пустым значением
        ArrayFill(body_fst,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(body_snd,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(shadow_fst,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(shadow_snd,start,count,INDICATOR_EMPTY_VALUE);
    }
}

//+-----+
//| Формирование основной части свечи
//+-----+
void FormMain(double &fst[],double &snd[],const double fst_value,
              const double snd_value,const int start,const int count)

```

```
{  
//--- проверка  
if(count!=0)  
{  
//--- заполняем индикаторные буферы значениями  
ArrayFill(fst,start,count,fst_value);  
ArrayFill(snd,start,count,snd_value);  
}  
}  
//+-----+  
// | Формирование конца свечи |  
//+-----+  
void FormEnd(double &fst[],double &snd[],const double fst_value,  
             const double snd_value,const int last)  
{  
//--- заполняем индикаторные буферы значениями  
ArrayFill(fst,last-1,2,fst_value);  
ArrayFill(snd,last-1,2,snd_value);  
}  
//+-----+  
// | Заполнение i-ого элемента индикаторных буферов пустыми значениями |  
//+-----+  
void FillIndicatorBuffers(const int i)  
{  
//--- устанавливаем пустое значение в ячейку индикаторных буферов  
ExtBearBodyFirst[i]=INDICATOR_EMPTY_VALUE;  
ExtBearBodySecond[i]=INDICATOR_EMPTY_VALUE;  
ExtBearShadowFirst[i]=INDICATOR_EMPTY_VALUE;  
ExtBearShadowSecond[i]=INDICATOR_EMPTY_VALUE;  
ExtBearBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;  
ExtBearBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;  
ExtBearShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;  
ExtBearShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;  
ExtBullBodyFirst[i]=INDICATOR_EMPTY_VALUE;  
ExtBullBodySecond[i]=INDICATOR_EMPTY_VALUE;  
ExtBullShadowFirst[i]=INDICATOR_EMPTY_VALUE;  
ExtBullShadowSecond[i]=INDICATOR_EMPTY_VALUE;  
ExtBullBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;  
ExtBullBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;  
ExtBullShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;  
ExtBullShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;  
}
```

## ArrayMinimum

Ищет минимальный элемент в первом измерении многомерного числового массива.

```
int ArrayMinimum(
    const void& array[],           // массив для поиска
    int start=0,                  // с какого индекса начинаем поиск
    int count=WHOLE_ARRAY         // количество проверяемых
);
```

### Параметры

*array[]*

[in] Числовой массив, в котором производится поиск.

*start=0*

[in] Начальный индекс для поиска.

*count=WHOLE\_ARRAY*

[in] Количество элементов для поиска. По умолчанию, ищет во всем массиве (*count=WHOLE\_ARRAY*).

### Возвращаемое значение

Функция возвращает индекс найденного элемента с учетом [серийности](#) массива. В случае неудачи функция возвращает -1.

### Примечание

Поиск минимального элемента производится с учетом значения флага [AS\\_SERIES](#).

Функции `ArrayMaximum` и `ArrayMinimum` принимают в качестве параметра массив любой размерности, при этом поиск происходит только по первому (нулевому) измерению.

### Пример:

```
#property description "Индикатор отображает свечи со старшего таймфрейма на текущем."
//--- настройки индикатора
#property indicator_chart_window
#property indicator_buffers 16
#property indicator_plots 8
//--- plot 1
#property indicator_label1 "BearBody"
#property indicator_color1 clrSeaGreen,clrSeaGreen
//--- plot 2
#property indicator_label2 "BearBodyEnd"
#property indicator_color2 clrSeaGreen,clrSeaGreen
//--- plot 3
#property indicator_label3 "BearShadow"
#property indicator_color3 clrSalmon,clrSalmon
//--- plot 4
#property indicator_label4 "BearShadowEnd"
#property indicator_color4 clrSalmon,clrSalmon
```

```

//---- plot 5
#property indicator_label5 "BullBody"
#property indicator_color5 clrOlive,clrOlive
//---- plot 6
#property indicator_label6 "BullBodyEnd"
#property indicator_color6 clrOlive,clrOlive
//---- plot 7
#property indicator_label7 "BullShadow"
#property indicator_color7 clrSkyBlue,clrSkyBlue
//---- plot 8
#property indicator_label8 "BullShadowEnd"
#property indicator_color8 clrSkyBlue,clrSkyBlue
//--- предопределенная константа
#define INDICATOR_EMPTY_VALUE 0.0
//--- входные параметры
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H4; // Таймфрейм для расчета индикатора
input datetime InpDateStart=D'2013.01.01 00:00'; // Дата начала анализа
//--- индикаторные буферы для медвежьих свечей
double ExtBearBodyFirst[];
double ExtBearBodySecond[];
double ExtBearBodyEndFirst[];
double ExtBearBodyEndSecond[];
double ExtBearShadowFirst[];
double ExtBearShadowSecond[];
double ExtBearShadowEndFirst[];
double ExtBearShadowEndSecond[];
//--- индикаторные буферы для бычьих свечей
double ExtBullBodyFirst[];
double ExtBullBodySecond[];
double ExtBullBodyEndFirst[];
double ExtBullBodyEndSecond[];
double ExtBullShadowFirst[];
double ExtBullShadowSecond[];
double ExtBullShadowEndFirst[];
double ExtBullShadowEndSecond[];
//--- глобальные переменные
datetime ExtTimeBuff[]; // буфер для времени с высшего таймфрейма
int ExtSize=0; // размер буфера времени
int ExtCount=0; // индекс внутри буфера времени
int ExtStartPos=0; // начальная позиция для расчета индикатора
bool ExtStartFlag=true; // вспомогательный флаг для получения начальной позиции
datetime ExtCurrentTime[1]; // последнее время формирования бара со старшего таймфрейма
datetime ExtLastTime; // последнее время со старшего таймфрейма, для которого индикатор
bool ExtBearFlag=true; // флаг для определения порядка записи данных в медвежьи индикаторы
bool ExtBullFlag=true; // флаг для определения порядка записи данных в бычьи индикаторы
int ExtIndexMax=0; // индекс максимального элемента в массиве
int ExtIndexMin=0; // индекс минимального элемента в массиве
int ExtDirectionFlag=0; // направление движения цены у текущей свечи
//--- отступ между ценой открытия и закрытия свечи для правильной отрисовки

```

```

const double ExtEmptyBodySize=0.2*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+-----+
//| Закрашивание основной части свечи |
//+-----+
void FillCandleMain(const double &open[],const double &close[],
                     const double &high[],const double &low[],
                     const int start,const int last,const int fill_index,
                     int &index_max,int &index_min)
{
//--- найдем индексы максимального и минимального элементов в массивах
    index_max=ArrayMaximum(high,ExtStartPos,last-start+1); // максимум в High
    index_min=ArrayMinimum(low,ExtStartPos,last-start+1); // минимум в Low
//--- определим сколько баров с текущего таймфрейма будем закрашивать
    int count=fill_index-start+1;
//--- если цена закрытия на первом баре больше цены закрытия на последнем - свеча медвежья
    if(open[start]>close[last])
    {
//--- если до этого свеча была бычьей, то очищаем значения бычьих индикаторных буферов
        if(ExtDirectionFlag!=-1)
            ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond);
//--- медвежья свеча
        ExtDirectionFlag=-1;
//--- формируем свечку
        FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,
                       close[last],high[index_max],low[index_min],start,count,ExtBearFlag);
//--- выход из функции
        return;
    }
//--- если цена закрытия на первом баре меньше цены закрытия на последнем - свеча бычья
    if(open[start]<close[last])
    {
//--- если до этого свеча была медвежьей, то очищаем значения медвежьих индикаторных буферов
        if(ExtDirectionFlag!=1)
            ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond);
//--- бычья свеча
        ExtDirectionFlag=1;
//--- формируем свечку
        FormCandleMain(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond,
                       open[start],high[index_max],low[index_min],start,count,ExtBullFlag);
//--- выход из функции
        return;
    }
//--- если попали в эту часть функции, то значит цена открытия на первом баре равняется
//--- цене закрытия на последнем баре; будем считать такую свечу медвежьей
//--- если до этого свеча была бычьей, то очищаем значения бычьих индикаторных буферов
    if(ExtDirectionFlag!=-1)
        ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond);
//--- медвежья свеча
    ExtDirectionFlag=-1;
}

```

```

//--- если цены закрытия и цены открытия равны, то используем сдвиг для корректного отображения
if(high[index_max]!=low[index_min])
    FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,
                    open[start]-ExtEmptyBodySize,high[index_max],low[index_min],start,fill_index);
else
    FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,
                    open[start],open[start]-ExtEmptyBodySize,high[index_max],
                    high[index_max]-ExtEmptyBodySize,start,count,ExtBearFlag);
}

//+-----+
//| Закрашивание конца свечи
//+-----+
void FillCandleEnd(const double &open[],const double &close[],
                    const double &high[],const double &low[],
                    const int start,const int last,const int fill_index,
                    const int index_max,const int index_min)
{
//--- если всего один бар, то не рисуем
if(last-start==0)
    return;

//--- если цена закрытия на первом баре больше цены закрытия на последнем - свеча медвежья
if(open[start]>close[last])
{
    //--- формируем конец свечи
    FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,ExtBearShadowEndSecond,
                  open[start],close[last],high[index_max],low[index_min],fill_index);
    //--- выход из функции
    return;
}

//--- если цена закрытия на первом баре меньше цены закрытия на последнем - свеча бычьего
if(open[start]<close[last])
{
    //--- формируем конец свечи
    FormCandleEnd(ExtBullBodyEndFirst,ExtBullBodyEndSecond,ExtBullShadowEndFirst,ExtBullShadowEndSecond,
                  close[last],open[start],high[index_max],low[index_min],fill_index);
    //--- выход из функции
    return;
}

//--- если попали в эту часть функции, то значит цена открытия на первом баре равняется
//--- цене закрытия на последнем баре; будем считать такую свечу медвежьей
//--- формируем конец свечи
if(high[index_max]!=low[index_min])
    FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,ExtBearShadowEndSecond,
                  open[start]-ExtEmptyBodySize,high[index_max],low[index_min],fill_index);
else
    FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,ExtBearShadowEndSecond,
                  open[start]-ExtEmptyBodySize,high[index_max],high[index_max]-ExtEmptyBodySize,fill_index);
}
//+-----+

```

```

//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- проверка периода индикатора
    if(!CheckPeriod((int)Period(),(int)InpPeriod))
        return(INIT_PARAMETERS_INCORRECT);
//--- отображение ценовых данных на переднем плане
    ChartSetInteger(0,CHART_FOREGROUND,0,1);
//--- привязка индикаторных буферов
    SetIndexBuffer(0,ExtBearBodyFirst);
    SetIndexBuffer(1,ExtBearBodySecond);
    SetIndexBuffer(2,ExtBearBodyEndFirst);
    SetIndexBuffer(3,ExtBearBodyEndSecond);
    SetIndexBuffer(4,ExtBearShadowFirst);
    SetIndexBuffer(5,ExtBearShadowSecond);
    SetIndexBuffer(6,ExtBearShadowEndFirst);
    SetIndexBuffer(7,ExtBearShadowEndSecond);
    SetIndexBuffer(8,ExtBullBodyFirst);
    SetIndexBuffer(9,ExtBullBodySecond);
    SetIndexBuffer(10,ExtBullBodyEndFirst);
    SetIndexBuffer(11,ExtBullBodyEndSecond);
    SetIndexBuffer(12,ExtBullShadowFirst);
    SetIndexBuffer(13,ExtBullShadowSecond);
    SetIndexBuffer(14,ExtBullShadowEndFirst);
    SetIndexBuffer(15,ExtBullShadowEndSecond);

//--- зададим некоторые значения свойств для построения индикатора
    for(int i=0;i<8;i++)
    {
        PlotIndexSetInteger(i,PLOT_DRAW_TYPE,DRAW_FILLING); // тип графического построения
        PlotIndexSetInteger(i,PLOT_LINE_STYLE,STYLE_SOLID); // стиль линии отрисовки
        PlotIndexSetInteger(i,PLOT_LINE_WIDTH,1);           // толщина линии отрисовки
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])

```

```

{
//--- если еще нет рассчитанных баров
    if(prev_calculated==0)
    {
        //--- получим время появления баров со старшего таймфрейма
        if(!GetTimeData())
            return(0);
    }

//--- установим прямое направление индексации
    ArraySetAsSeries(time, false);
    ArraySetAsSeries(high, false);
    ArraySetAsSeries(low, false);
    ArraySetAsSeries(open, false);
    ArraySetAsSeries(close, false);

//--- переменная начала для расчета баров
    int start=prev_calculated;
//--- если бар формируется, то пересчитываем значение индикатора на нем
    if(start!=0 && start==rates_total)
        start--;
//--- цикл расчета значений индикатора
    for(int i=start;i<rates_total;i++)
    {
        //--- заполняем i-ые элементы индикаторных буферов пустыми значениями
        FillIndicatorBuffers(i);
        //--- проводим вычисление для баров начиная с даты InpDateStart
        if(time[i]>=InpDateStart)
        {
            //--- в первый раз определим позицию с которой начнем отображать значения
            if(ExtStartFlag)
            {
                //--- запомним номер начального бара
                ExtStartPos=i;
                //--- определим первую дату со старшего таймфрейма, которая больше time[i]
                while(time[i]>=ExtTimeBuff[ExtCount])
                    if(ExtCount<ExtSize-1)
                        ExtCount++;
                //--- изменим значение флага, чтобы больше не попадать в этот блок
                ExtStartFlag=false;
            }
            //--- проверка, есть ли еще в массиве элементы
            if(ExtCount<ExtSize)
            {
                //--- ждем, когда значение времени с текущего таймфрейма достигнет значени
                if(time[i]>=ExtTimeBuff[ExtCount])
                {
                    //--- рисуем главную часть свечи (без закрашивания между последним и пр
                    FillCandleMain(open,close,high,low,ExtStartPos,i-1,i-2,ExtIndexMax,Ext
                    //--- закрашиваем конец свечи (область между последним и предпоследним
                    FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,Ext
                }
            }
        }
    }
}

```

```

        //--- сдвигаем начальную позицию для рисования следующей свечи
ExtStartPos=i;
//--- увеличиваем счетчик массива
ExtCount++;
}
else
    continue;
}
else
{
//--- сбрасываем значения ошибки
ResetLastError();
//--- получаем последнюю дату со старшего таймфрейма
if(CopyTime(Symbol(),InpPeriod,0,1,ExtCurrentTime)==-1)
{
Print("Ошибка копирования данных, код = ",GetLastError());
return(0);
}
//--- если новая дата больше, то значит заканчиваем формирование свечки
if(ExtCurrentTime[0]>ExtLastTime)
{
//--- очистим область между последним и предпоследним баром в главных индикаторах
ClearEndOfBodyMain(i-1);
//--- закрасим эту область с помощью вспомогательных индикаторных буферов
FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtIndexMin);
//--- сдвигаем начальную позицию для рисования следующей свечи
ExtStartPos=i;
//--- сбросим флаг направления цены
ExtDirectionFlag=0;
//--- запомним новую последнюю дату
ExtLastTime=ExtCurrentTime[0];
}
else
{
//--- формируем свечку
FillCandleMain(open,close,high,low,ExtStartPos,i,i,ExtIndexMax,ExtIndexMin);
}
}
}
}
}

//--- return value of prev_calculated for next call
return(rates_total);
}

//+-----+
//| Проверка введенного периода индикатора на корректность |
//+-----+
bool CheckPeriod(int current_period,int high_period)
{
//--- период индикатора должен быть больше таймфрейма, на котором он отображается
}

```

```

if(current_period>=high_period)
{
    Print("Ошибка! Значение периода индикатора должно быть больше значения текущего");
    return(false);
}

//--- если период индикатора - одна неделя или месяц, то период корректен
if(high_period>32768)
    return(true);

//--- приведем значения периодов к минутам
if(high_period>30)
    high_period=(high_period-16384)*60;
if(current_period>30)
    current_period=(current_period-16384)*60;

//--- период индикатора должен быть кратен таймфрейму, на котором он отображается
if(high_period%current_period!=0)
{
    Print("Ошибка! Значение периода индикатора должно быть кратным значению текущего таймфрейма");
    return(false);
}

//--- период индикатора должен превышать значение таймфрейма, на котором он отображается
if(high_period/current_period<3)
{
    Print("Ошибка! Значение периода индикатора должно превышать значение текущего таймфрейма");
    return(false);
}

//--- период индикатора корректен для текущего таймфрейма
return(true);
}

//-----+
//| Получение данных времени со старшего таймфрейма |+
//-----+



bool GetTimeData(void)
{
//--- сброс значения ошибки
ResetLastError();

//--- скопируем все данные на текущее время
if(CopyTime(Symbol(), InpPeriod, InpDateStart, TimeCurrent(), ExtTimeBuff)==-1)
{
    //--- получим код ошибки
    int code=GetLastError();
    //--- распечатаем текст ошибки
    PrintFormat("Ошибка копирования данных! %s", code==4401
        ? "История еще подгружается!"
        : "Код = "+IntegerToString(code));
    //--- вернем false для повторной попытки загрузки данных
    return(false);
}
//--- получим размер массива
ExtSize=ArraySize(ExtTimeBuff);
}

```

```

//--- установим индекс цикла для массива равным нулю
ExtCount=0;
//--- установим позицию текущей свечи на данном таймфрейме равной нулю
ExtStartPos=0;
ExtStartFlag=true;
//--- запомним последнее значение времени со старшего таймфрейма
ExtLastTime=ExtTimeBuff[ExtSize-1];
//--- успешное выполнение
return(true);
}

//+-----+
//| Функция формирует основную часть свечи. В зависимости от значения |
//| флага, функция определяет, какие данные в какие массивы должны     |
//| записываться для корректного отображения.                           |
//+-----+
void FormCandleMain(double &body_fst[],double &body_snd[],
                     double &shadow_fst[],double &shadow_snd[],
                     const double fst_value,const double snd_value,
                     const double fst_extremum,const double snd_extremum,
                     const int start,const int count,const bool flag)
{
//--- проверяем значение флага
if(flag)
{
    //--- формируем тело свечи
    FormMain(body_fst,body_snd,fst_value,snd_value,start,count);
    //--- формируем тень свечи
    FormMain(shadow_fst,shadow_snd,fst_extremum,snd_extremum,start,count);
}
else
{
    //--- формируем тело свечи
    FormMain(body_fst,body_snd,snd_value,fst_value,start,count);
    //--- формируем тень свечи
    FormMain(shadow_fst,shadow_snd,snd_extremum,fst_extremum,start,count);
}
}

//+-----+
//| Функция формирует конец свечи. В зависимости от значения флага,   |
//| функция определяет, какие данные в какие массивы должны             |
//| записываться для корректного отображения.                           |
//+-----+
void FormCandleEnd(double &body_fst[],double &body_snd[],
                    double &shadow_fst[],double &shadow_snd[],
                    const double fst_value,const double snd_value,
                    const double fst_extremum,const double snd_extremum,
                    const int end,bool &flag)
{
//--- проверяем значение флага
}

```

```

if(flag)
{
    //--- формируем конец тела свечи
    FormEnd(body_fst,body_snd,fst_value,snd_value,end);
    //--- формируем конец тени свечи
    FormEnd(shadow_fst,shadow_snd,fst_extremum,snd_extremum,end);
    //--- меняем значение флага на противоположное
    flag=false;
}
else
{
    //--- формируем конец тела свечи
    FormEnd(body_fst,body_snd,snd_value,fst_value,end);
    //--- формируем конец тени свечи
    FormEnd(shadow_fst,shadow_snd,snd_extremum,fst_extremum,end);
    //--- меняем значение флага на противоположное
    flag=true;
}

//+-----+
//| Очистка конца свечи (область между последним и предпоследним
//| баром)
//+-----+
void ClearEndOfBodyMain(const int ind)
{
    ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSecond,ind);
    ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSecond,ind);
}

//+-----+
//| Очистка свечи
//+-----+
void ClearCandle(double &body_fst[],double &body_snd[],double &shadow_fst[],
                 double &shadow_snd[],const int start,const int count)
{
    //--- проверка
    if(count!=0)
    {
        //--- заполняем индикаторные буфера пустым значением
        ArrayFill(body_fst,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(body_snd,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(shadow_fst,start,count,INDICATOR_EMPTY_VALUE);
        ArrayFill(shadow_snd,start,count,INDICATOR_EMPTY_VALUE);
    }
}

//+-----+
//| Формирование основной части свечи
//+-----+
void FormMain(double &fst[],double &snd[],const double fst_value,
              const double snd_value,const int start,const int count)

```

```

{
//--- проверка
if(count!=0)
{
    //--- заполняем индикаторные буферы значениями
    ArrayFill(fst,start,count,fst_value);
    ArrayFill(snd,start,count,snd_value);
}
}

//+-----+
//| Формирование конца свечи |
//+-----+
void FormEnd(double &fst[],double &snd[],const double fst_value,
             const double snd_value,const int last)
{
    //--- заполняем индикаторные буферы значениями
    ArrayFill(fst,last-1,2,fst_value);
    ArrayFill(snd,last-1,2,snd_value);
}

//+-----+
//| Заполнение i-ого элемента индикаторных буферов пустыми значениями|
//+-----+
void FillIndicatorBuffers(const int i)
{
    //--- устанавливаем пустое значение в ячейку индикаторных буферов
    ExtBearBodyFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodySecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBearShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodySecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
    ExtBullShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
}
}

```

## ArrayPrint

Выводит в журнал массив простого типа или простой структуры.

```
void ArrayPrint(
    const void& array[],           // выводимый массив
    uint digits=_Digits,           // количество десятичных знаков после запятой
    const string separator=NULL,   // разделитель между значениями полей структуры
    ulong start=0,                // индекс первого выводимого элемента
    ulong count=WHOLE_ARRAY,      // количество выводимых элементов
    ulong flags=ARRAYPRINT_HEADER|ARRAYPRINT_INDEX|ARRAYPRINT_LIMIT|ARRAYPRINT_ALIGN
);
```

### Параметры

*array[]*

[in] Массив простого типа или [простой структуры](#).

*digits=\_Digits*

[in] Количество знаков после запятой для вещественных типов. По умолчанию равно [\\_Digits](#).

*separator=NULL*

[in] Разделитель между значениями полей элемента структуры. Значение по умолчанию [NULL](#) означает пустую строку, в этом случае разделителем является пробел.

*start=0*

[in] Индекс первого выводимого элемента массива. По умолчанию выводится с нулевого индекса.

*count=WHOLE\_ARRAY*

[in] Количество элементов массива, которые нужно вывести. По умолчанию выводится весь массив (*count=WHOLE\_ARRAY*).

*flags=ARRAYPRINT\_HEADER|ARRAYPRINT\_INDEX|ARRAYPRINT\_LIMIT|ARRAYPRINT\_ALIGN*

[in] Комбинация флагов, задающая режим вывода. По умолчанию включены все флаги:

- ARRAYPRINT\_HEADER** - вывод заголовков для массива структур
- ARRAYPRINT\_INDEX** - вывод слева номера индекса
- ARRAYPRINT\_LIMIT** - вывод только 100 первых и 100 последних элементов массива.  
Используется, если нужно вывести только часть большого массива.
- ARRAYPRINT\_ALIGN** - включить выравнивание выводимых значений - числа будут выравниваться вправо, строки влево.
- ARRAYPRINT\_DATE** - при выводе datetime выводить дату в формате dd.mm.yyyy
- ARRAYPRINT\_MINUTES** - при выводе datetime выводить время в формате HH:MM
- ARRAYPRINT\_SECONDS** - при выводе datetime выводить время в формате HH:MM:SS

### Возвращаемое значение

Нет

### Примечание

ArrayPrint() выводит в журнал не все поля массива структур - поля-массивы и поля-[указатели объектов](#) пропускаются. Эти столбцы просто не будут выведены на печать для поддержания

простого и удобного представления. Если вам нужен вывод всех полей такой структуры, то вам необходимо написать собственную функцию массового вывода с желаемым форматированием.

### Пример:

```
//--- выводит значения 10 последних баров
MqlRates rates[];
if(CopyRates(_Symbol,_Period,1,10,rates))
{
    ArrayPrint(rates);
    Print("Проверка\n[time]\t[open]\t[high]\t[low]\t[close]\t[tick_volume]\t[spread]");
    for(int i=0;i<10;i++)
    {
        PrintFormat("%d\t%s\t%G\t%G\t%G\t%G\t%I64d\t",i,
TimeToString(rates[i].time,TIME_DATE|TIME_MINUTES|TIME_SECONDS),
rates[i].open,rates[i].high,rates[i].low,rates[i].close,
rates[i].tick_volume,rates[i].spread,rates[i].real_volume);
    }
}
else
    PrintFormat("CopyRates failed, error code=%d",GetLastError());
//--- пример вывода
/*
[time] [open] [high] [low] [close] [tick_volume] [spread] [real_volume]
[0] 2016.11.09 04:00:00 1.11242 1.12314 1.11187 1.12295 18110 10 17300175000
[1] 2016.11.09 05:00:00 1.12296 1.12825 1.11930 1.12747 17829 9 15632176000
[2] 2016.11.09 06:00:00 1.12747 1.12991 1.12586 1.12744 13458 10 9593492000
[3] 2016.11.09 07:00:00 1.12743 1.12763 1.11988 1.12194 15362 9 12352245000
[4] 2016.11.09 08:00:00 1.12194 1.12262 1.11058 1.11172 16833 9 12961333000
[5] 2016.11.09 09:00:00 1.11173 1.11348 1.10803 1.11052 15933 8 10720384000
[6] 2016.11.09 10:00:00 1.11052 1.11065 1.10289 1.10528 11888 9 8084811000
[7] 2016.11.09 11:00:00 1.10512 1.11041 1.10472 1.10915 7284 10 5087113000
[8] 2016.11.09 12:00:00 1.10915 1.11079 1.10892 1.10904 8710 9 6769629000
[9] 2016.11.09 13:00:00 1.10904 1.10913 1.10223 1.10263 8956 7 7192138000
*/

```

### Смотри также

[FileSave](#), [FileLoad](#)

## ArrayRange

Возвращает число элементов в указанном измерении массива.

```
int ArrayRange(
    const void& array[],           // массив для проверки
    int rank_index                 // номер измерения
);
```

### Параметры

*array[]*  
 [in] Проверяемый массив.  
*rank\_index*  
 [in] Индекс измерения.

### Возвращаемое значение

Число элементов в указанном измерении массива

### Примечание

Поскольку индексы начинаются с нуля, количество измерений массива на единицу больше, чем индекс последнего измерения.

### Пример:

```
void OnStart()
{
//--- создадим четырехмерный массив
double array[][5][2][4];
//--- зададим размер нулевого измерения
ArrayResize(array,10,10);
//--- распечатаем размерности измерений
int temp;
for(int i=0;i<4;i++)
{
//--- получим размер i-ого измерения
temp=ArrayRange(array,i);
//--- распечатаем
PrintFormat("dim = %d, range = %d",i,temp);
}
//--- Результат
// dim = 0, range = 10
// dim = 1, range = 5
// dim = 2, range = 2
// dim = 3, range = 4
}
```

## ArrayResize

Устанавливает новый размер в первом измерении массива

```
int ArrayResize(
    void& array[],           // массив, переданный по ссылке
    int new_size,             // новый размер массива
    int reserve_size=0        // резервное значение размера (избыточное)
);
```

### Параметры

*array[]*

[out] Массив для изменения размеров.

*new\_size*

[in] Новый размер для первого измерения.

*reserve\_size=0*

[in] Размер для дополнительного резерва.

### Возвращаемое значение

При успешном выполнении функция возвращает количество всех элементов, содержащихся в массиве после изменения размера; в противном случае возвращает -1 и массив не меняет размеры.

Если `ArrayResize()` применена к [статическому](#) массиву, [таймсерии](#) или [индикаторному буферу](#), то размер массива остается прежним - такие массивы не могут быть перераспределены. В этом случае если `new_size<=ArraySize(array)`, то функция просто вернет `new_size`; в противном случае будет возвращено -1.

### Примечание

Функция может быть применена только к [динамическим массивам](#). При этом необходимо иметь ввиду, что нельзя изменять размер для динамических массивов, назначенных в качестве индикаторных буферов функцией [SetIndexBuffer\(\)](#). Для индикаторных буферов все операции по изменению размера производит исполняющая подсистема терминала.

Общее число элементов в массиве не может превышать 2147483647.

При частом распределении памяти рекомендуется использовать третий параметр, задающий резерв для уменьшения количества физического распределения памяти. Все последующие вызовы функции `ArrayResize` не приводят к физическому перераспределению памяти, а только меняется размер первого измерения массива в пределах зарезервированной памяти. Следует помнить, что третий параметр будет использоваться только тогда, когда будет происходить физическое распределение памяти, например:

```
ArrayResize(arr,1000,1000);
for(int i=1;i<3000;i++)
    ArrayResize(arr,i,1000);
```

В данном случае произойдёт 2 перераспределения памяти, один раз до входа в цикл на 3000 итераций, при этом размерность массива будет установлена в 1000 и второй при *i* равной 2000.

Если третий параметр опустить, то произойдёт 2000 физических перераспределения памяти и это замедлит выполнение программы.

**Пример:**

```
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- счетчики
    ulong start=GetTickCount();
    ulong now;
    int count=0;
//--- массив для демонстрации быстрого варианта
    double arr[];
    ArrayResize(arr,100000,100000);
//--- проверим, как быстро работает вариант с резервированием памяти
    Print("--- Test Fast: ArrayResize(arr,100000,100000));"
    for(int i=1;i<=300000;i++)
    {
//--- задаем новый размер массива с указанием резерва в 100000 элементов!
        ArrayResize(arr,i,100000);
//--- при достижении круглой цифры выводим размер массива и затраченное время
        if(ArraySize(arr)%100000==0)
        {
            now=GetTickCount();
            count++;
            PrintFormat("%d. ArraySize(arr)=%d Time=%d ms",count,ArraySize(arr),(now-start));
            start=now;
        }
    }
//--- покажем теперь, как медленно работает вариант без резервирования памяти
    double slow[];
    ArrayResize(slow,100000,100000);
//---
    count=0;
    start=GetTickCount();
    Print("---- Test Slow: ArrayResize(slow,100000));"
//---
    for(int i=1;i<=300000;i++)
    {
//--- задаем новый размер массива, но уже без дополнительного резерва
        ArrayResize(slow,i);
//--- при достижении круглой цифры выводим размер массива и затраченное время
        if(ArraySize(slow)%100000==0)
        {
            now=GetTickCount();
            count++;
        }
    }
}
```

```
        PrintFormat ("%d. ArraySize(slow)=%d Time=%d ms", count, ArraySize(slow), (now-start));
        start=now;
    }
}
//--- Примерный результат выполнения скрипта
/*
Test_ArrayResize (EURUSD,H1)    --- Test Fast: ArrayResize(arr,100000,100000)
Test_ArrayResize (EURUSD,H1)    1. ArraySize(arr)=100000 Time=0 ms
Test_ArrayResize (EURUSD,H1)    2. ArraySize(arr)=200000 Time=0 ms
Test_ArrayResize (EURUSD,H1)    3. ArraySize(arr)=300000 Time=0 ms
Test_ArrayResize (EURUSD,H1)    ---- Test Slow: ArrayResize(slow,100000)
Test_ArrayResize (EURUSD,H1)    1. ArraySize(slow)=100000 Time=0 ms
Test_ArrayResize (EURUSD,H1)    2. ArraySize(slow)=200000 Time=0 ms
Test_ArrayResize (EURUSD,H1)    3. ArraySize(slow)=300000 Time=228511 ms
*/

```

**Смотри также**[ArrayInitialize](#)

## ArrayInsert

Вставляет в массив-приемник из массива-источника указанное число элементов, начиная с указанного индекса.

```
bool ArrayInsert(
    void&      dst_array[],           // массив-приемник
    const void& src_array[],          // массив источник
    uint        dst_start,            // индекс в массиве-приемнике для вставки
    uint        src_start=0,           // индекс в массиве-источнике для копирования
    uint        count=WHOLE_ARRAY     // количество вставляемых элементов
);
```

### Параметры

*dst\_array[]*

[in][out] Массив-приемник, в который необходимо добавить элементы.

*src\_array[]*

[in] Массив-источник, из которого необходимо добавить элементы.

*dst\_start*

[in] Индекс в массиве-приемнике для вставки элементов из массива-источника.

*src\_start=0*

[in] Индекс в массиве-приемнике, начиная с которого берутся элементы массива-источника для вставки.

*count=WHOLE\_ARRAY*

[in] Количество добавляемых элементов из массива-источника. Значение [WHOLE\\_ARRAY](#) означает все элементы с указанного индекса до конца массива.

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 5052 - `ERR_SMALL_ARRAY` (параметры `start` и/или `count` заданы неверно или исходный массив `src_array[]` является пустым),
- 5056 - `ERR_SERIES_ARRAY` (массив не может быть изменен, индикаторный буфер),
- 4006 - `ERR_INVALID_ARRAY` (копирование в себя недопустимо, либо массивы имеют разный тип, либо массив фиксированного размера, который содержит объекты класса или структуры с деструктором),
- 4005 - `ERR_STRUCT_WITHOBJECTS_ORCLASS` (массив содержит не [POD-структуры](#), то есть простое копирование невозможно),
- ошибки изменения размера массива-приемника `dst_array[]` - они приведены в описании функции [ArrayRemove\(\)](#).

### Примечание

Если функция используется для массива фиксированного размера, то сам размер массива-приемника `dst_array[]` не меняется, при этом начиная с позиции `dst_start` элементы массива-

приемника сдвигаются вправо (последние `count` элементов "выпадают"), а на освободившееся место происходит копирование элементов из массива-источника.

Нельзя вставлять элементы в динамические массивы, назначенные в качестве индикаторных буферов функцией `SetIndexBuffer()`. Для индикаторных буферов все операции по изменению размера производят исполняющая подсистема терминала.

В массиве-источнике элементы копируются, начиная с индекса `src_start`. Размер массива-источника при этом не изменяется. Добавляемые в массив-приемник элементы не являются ссылками на элементы массива-источника - это означает, что последующие изменения элементов в любом из двух массивов не отражаются на втором.

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- объявим массив фиксированного размера и заполним значениями
int array_dest[10];
for(int i=0;i<10;i++)
{
    array_dest[i]=i;
}
//--- массив-источник
int array_source[10];
for(int i=0;i<10;i++)
{
    array_source[i]=10+i;
}
//--- покажем массивы до вставки элементов
Print("До вызова ArrayInsert()");
ArrayPrint(array_dest);
ArrayPrint(array_source);
//--- вставим 3 элемента из массива-источника и покажем новый состав массив-приемника
ArrayInsert(array_dest,array_source,4,0,3);
Print("После вызова ArrayInsert()");
ArrayPrint(array_dest);
/*
Результат выполнения
До вызова ArrayInsert()
0 1 2 3 4 5 6 7 8 9
После вызова ArrayInsert()
0 1 2 3 10 11 12 7 8 9
*/
}
```

### Смотри также

[ArrayRemove](#), [ArrayCopy](#), [ArrayResize](#), [ArrayFree](#)



## ArrayRemove

Удаляет из массива указанное число элементов начиная с указанного индекса.

```
bool ArrayRemove(
    void& array[],           // массив любого типа
    uint start,              // с какого индекса начинаем удалять
    uint count=WHOLE_ARRAY   // количество элементов
);
```

### Параметры

*array[]*

[in][out] Массив.

*start*

[in] Индекс, начиная с которого удаляются элементы массива.

*count=WHOLE\_ARRAY*

[in] Количество удаляемых элементов. Значение [WHOLE\\_ARRAY](#) означает удаление всех элементов с указанного индекса до конца массива.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 5052 - ERR\_SMALL\_ARRAY (значение *start* слишком большое),
- 5056 - ERR\_SERIES\_ARRAY (массив не может быть изменен, индикаторный буфер),
- 4003 - ERR\_INVALID\_PARAMETER (значение *count* слишком большое),
- 4005 - ERR\_STRUCT\_WITHOBJECTS\_ORCLASS (массив фиксированного размера, который содержит сложные объекты с деструктором),
- 4006 - ERR\_INVALID\_ARRAY (массив фиксированного размера, который содержит объекты структур или классов с деструктором).

### Примечание

Если функция используется для массива фиксированного размера, то сам размер массива не меняется: при этом происходит физическое копирование оставшегося "хвоста" в позицию *start*. Для точного понимания работы функциисмотрите пример ниже. "Физическое" копирование означает, что копируемые объекты не создаются с помощью вызова конструктора или оператора копирования, а просто происходит копирование бинарного представления объекта. Именно по этой причине запрещается применять функцию `ArrayRemove()` к массиву фиксированного размера, содержащего объекты с деструктором (вводится ошибка `ERR_INVALID_ARRAY` или `ERR_STRUCT_WITHOBJECTS_ORCLASS`). Так как при удалении такого объекта деструктор должен быть вызван дважды - для первоначального объекта и его копии.

Нельзя удалять элементы из динамических массивов, назначенных в качестве индикаторных буферов функцией [SetIndexBuffer\(\)](#), это приведет к появлению ошибки `ERR_SERIES_ARRAY`. Для индикаторных буферов все операции по изменению размера производит исполняющая подсистема терминала.

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- объявим массив фиксированного размера и заполним значениями
int array[10];
for(int i=0;i<10;i++)
{
    array[i]=i;
}
//--- покажем массив до удаления элементов
Print("До вызова ArrayRemove()");
ArrayPrint(array);
//--- удалим 2 элемента из массива и покажем новый состав
ArrayRemove(array,4,2);
Print("После вызова ArrayRemove()");
ArrayPrint(array);
/*
Результат выполнения:
До вызова ArrayRemove()
0 1 2 3 4 5 6 7 8 9
После вызова ArrayRemove()
0 1 2 3 6 7 8 9 8 9
*/
}
```

#### Смотри также

[ArrayInsert](#), [ArrayCopy](#), [ArrayResize](#), [ArrayFree](#)

## ArrayReverse

Разворачивает в массиве указанное число элементов начиная с указанного индекса.

```
bool ArrayReverse(
    void& array[],           // массив любого типа
    uint start=0,             // с какого индекса начинаем переворачивать массив
    uint count=WHOLE_ARRAY    // количество элементов
);
```

### Параметры

*array[]*  
[in][out] Массив.

*start=0*  
[in] Индекс, начиная с которого разворачивается массив.

*count=WHOLE\_ARRAY*  
[in] Количество разворачиваемых элементов. Если указано значение WHOLE\_ARRAY, то будут зеркально перемещены между собой все элементы массива, начиная с указанного индекса *start* и до конца массива.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Функция [ArraySetAsSeries\(\)](#) физически не перемещает элементы массива, а только меняет направление индексации задом наперед для организации доступа к элементам как в [таймсерию](#). Функция `ArrayReverse()` физически перемещает элементы массива таким образом, что массив "переворачивается".

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- объявим массив фиксированного размера и заполним значениями
    int array[10];
    for(int i=0;i<10;i++)
    {
        array[i]=i;
    }
//--- покажем массив до разворачивания элементов
    Print("До вызова ArrayReverse()");
    ArrayPrint(array);
//--- развернем 3 элемента в массиве и покажем новый состав
    ArrayReverse(array,4,3);
```

```
Print("После вызова ArrayReverse() ");
ArrayPrint(array);
/*
Результат выполнения:
До вызова ArrayReverse()
0 1 2 3 4 5 6 7 8 9
После вызова ArrayReverse()
0 1 2 3 6 5 4 7 8 9
*/
```

#### Смотри также

[ArrayInsert](#), [ArrayRemove](#), [ArrayCopy](#), [ArrayResize](#), [ArrayFree](#), [ArrayGetAsSeries](#), [ArraySetAsSeries](#)

## ArraySetAsSeries

Устанавливает флаг AS\_SERIES указанному [объекту динамического массива](#), индексация элементов массива будет производиться как в [таймсерииях](#).

```
bool ArraySetAsSeries(
    const void& array[],      // массив по ссылке
    bool        flag         // true означает обратный порядок индексации
);
```

### Параметры

*array[]*

[in][out] Числовой массив для установки.

*flag*

[in] Направление индексирования массива.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Флаг [AS\\_SERIES](#) не может быть установлен у многомерных массивов и у статических массивов (то есть массивов, чей размер в квадратных скобках указан еще на этапе компиляции). Индексация в таймсериии отличается от обычного массива тем, что индексация элементов таймсериии производится от конца массива к началу (от самых свежих данных к самым старым).

### Пример: индикатор, показывающий номер бара



```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot Numeration
#property indicator_label1 "Numeration"
```

```

#property indicator_type1    DRAW_LINE
#property indicator_color1   CLR_NONE
//--- indicator buffers
double          NumerationBuffer[];
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,NumerationBuffer,INDICATOR_DATA);
//--- установим индексацию для буфера как в таймсерии
    ArraySetAsSeries(NumerationBuffer,true);
//--- установим точность отображения в DataWindow
    IndicatorSetInteger(INDICATOR_DIGITS,0);
//--- как будет выглядеть в DataWindow имя индикаторного массива
    PlotIndexSetString(0,PLOT_LABEL,"Bar #");
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function                |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//--- будем хранить время открытия текущего нулевого бара
    static datetime currentBarTimeOpen=0;
//--- перевернем доступ к массиву time[] - сделаем как в таймсерии
    ArraySetAsSeries(time,true);
//--- если время нулевого бара отличается от того, что мы храним
    if(currentBarTimeOpen!=time[0])
    {
//--- пронумеруем все бары от текущего момента вглубь графика
        for(int i=rates_total-1;i>0;i--) NumerationBuffer[i]=i;
        currentBarTimeOpen=time[0];
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}

```

Смотри также

[Доступ к таймсериям](#), [ArrayGetAsSeries](#)

## ArraySize

Возвращает количество элементов указанного массива.

```
int ArraySize(
    const void& array[] // проверяемый массив
);
```

### Параметры

*array[]*  
[in] Массив любого типа.

### Возвращаемое значение

Значение типа [int](#).

### Примечание

Для одномерного массива значение, возвращаемое функцией [ArraySize](#), равно значению [ArrayRange\(array,0\)](#).

### Пример:

```
void OnStart()
{
//--- создание массивов
    double one_dim[];
    double four_dim[][10][5][2];
//--- размеры
    int one_dim_size=25;
    int reserve=20;
    int four_dim_size=5;
//--- вспомогательная переменная
    int size;
//--- выделим память без резервирования
    ArrayResize(one_dim,one_dim_size);
    ArrayResize(four_dim,four_dim_size);
//--- 1. одномерный массив
    Print("=====+");
    Print("Размеры массивов:");
    Print("1. Одномерный массив");
    size=ArraySize(one_dim);
    PrintFormat("Размер нулевого измерения = %d, Размер массива = %d",one_dim_size,size);
//--- 2. многомерный массив
    Print("2. Многомерный массив");
    size=ArraySize(four_dim);
    PrintFormat("Размер нулевого измерения = %d, Размер массива = %d",four_dim_size,size);
//--- размеры измерений
    int d_1=ArrayRange(four_dim,1);
    int d_2=ArrayRange(four_dim,2);
    int d_3=ArrayRange(four_dim,3);
```

```
Print("Проверка:");
Print("Нулевое измерение = Размер массива / (Первое измерение * Второе измерение *
PrintFormat("%d = %d / (%d * %d * %d)",size/(d_1*d_2*d_3),size,d_1,d_2,d_3);
//--- 3. одномерный массив с резервированием памяти
Print("3. Одномерный массив с резервированием памяти");
//--- увеличим значение в 2 раза
one_dim_size*=2;
//--- выделим память с резервированием
ArrayResize(one_dim,one_dim_size,reserve);
//--- распечатаем размер
size=ArraySize(one_dim);
PrintFormat("Размер с резервированием = %d, Реальный размер массива = %d",one_dim_size);
}
```

## ArraySort

Сортирует многомерный числовой массив по возрастанию значений в первом измерении.

```
bool ArraySort(
    void& array[]          // массив для сортировки
);
```

### Параметры

*array[]*  
 [in][out] Числовой массив для сортировки.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Массив всегда сортируется по возрастанию, независимо от значения флага [AS\\_SERIES](#).

Функции `ArraySort` и `ArrayBSearch` принимают в качестве параметра массив любой размерности, при этом сортировка и поиск происходят только по первому (нулевому) измерению.

### Пример:

```
#property description "Индикатор анализирует данные за последний месяц и раскрашивает
#property description "и большими тиковыми объемами. Для определения таких свечей прои
#property description "массива тиковых объемов. Свечи, объемы которых составляют первые
#property description "процентов массива, считаются малыми. Свечи, тиковые объемы которы
#property description "последние InpBigVolume процентов массива, считаются большими."
//--- настройки индикатора
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots 1
//--- plot
#property indicator_label1 "VolumeFactor"
#property indicator_type1 DRAW_COLOR_CANDLES
#property indicator_color1 clrDodgerBlue,clrOrange
#property indicator_style1 STYLE_SOLID
#property indicator_width1 2
//--- предопределенная константа
#define INDICATOR_EMPTY_VALUE 0.0
//--- входные параметры
input int InpSmallVolume=15; // Процент малых объемов (<50)
input int InpBigVolume=80; // Процент больших объемов (<50)
//--- время начала анализа (будет смещаться)
datetime ExtStartTime;
//--- индикаторные буфера
double ExtOpenBuff[];
double ExtHighBuff[];
double ExtLowBuff[];
```

```

double ExtCloseBuff[];
double ExtColorBuff[];
//--- граничные значения объемов для отображения свечей
long ExtLeftBorder=0;
long ExtRightBorder=0;
//+-----+
//| Получение значений границ для тиковых объемов |+
//+-----+
bool GetVolumeBorders(void)
{
//--- переменные
    datetime stop_time; // время окончания копирования
    long buff[]; // буфер, куда будем копировать
//--- время окончания - текущее время
    stop_time=TimeCurrent();
//--- время начала - на месяц раньше от текущего
    ExtStartTime=GetStartTime(stop_time);
//--- получим значения тиковых объемов
    ResetLastError();
    if(CopyTickVolume(Symbol(),Period(),ExtStartTime,stop_time,buff)==-1)
    {
//--- не удалось получить данные, вернем false для запуска команды на пересчет
        PrintFormat("Не удалось получить значения тикового объема. Код ошибки = %d",GetLastError());
        return(false);
    }
//--- вычислим размер массива
    int size=ArraySize(buff);
//--- отсортируем массив
    ArraySort(buff);
//--- определим значения левой и правой границы для тиковых объемов
    ExtLeftBorder=buff[size*InpSmallVolume/100];
    ExtRightBorder=buff[(size-1)*(100-InpBigVolume)/100];
//--- успешное выполнение
    return(true);
}
//+-----+
//| Получение даты на месяц меньше чем дата во входном параметре |+
//+-----+
datetime GetStartTime(const datetime stop_time)
{
//--- преобразуем время окончания к переменной структуры типа MqlDateTime
    MqlDateTime temp;
    TimeToStruct(stop_time,temp);
//--- получим дату на месяц меньше
    if(temp.mon>1)
        temp.mon-=1; // текущий месяц не первый в году, значит номер предыдущего на один
    else
    {
        temp.mon=12; // текущий месяц первый в году, значит номер предыдущего равен 12
    }
}

```

```

        temp.year-=1; // а номер года на один меньше
    }
//--- число дня будет не больше 28
if(temp.day>28)
    temp.day=28;
//--- вернем полученную дату
return(StructToTime(temp));
}

//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- проверка, удовлетворяют ли входные параметры условиям
if(InpSmallVolume<0 || InpSmallVolume>=50 || InpBigVolume<0 || InpBigVolume>=50)
{
    Print("Некорректные входные параметры");
    return(INIT_PARAMETERS_INCORRECT);
}
//--- indicator buffers mapping
SetIndexBuffer(0,ExtOpenBuff);
SetIndexBuffer(1,ExtHighBuff);
SetIndexBuffer(2,ExtLowBuff);
SetIndexBuffer(3,ExtCloseBuff);
SetIndexBuffer(4,ExtColorBuff,INDICATOR_COLOR_INDEX);
//--- зададим значение, которое не будет отображаться
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VALUE);
//--- зададим надписи для индикаторных буферов
PlotIndexSetString(0,PLOT_LABEL,"Open;High;Low;Close");
//---
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function               |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- проверка, есть ли еще необработанные бары
if(prev_calculated<rates_total)
{
}

```

```

//--- получение новых значений левой и правой границы для объемов
if(!GetVolumeBorders())
    return(0);
}

//--- переменная начала для расчета баров
int start=prev_calculated;
//--- если значения индикатора уже были рассчитаны на предыдущем тике, то работаем на
if(start>0)
    start--;
//--- установим прямое направление индексации в таймсерииях
ArraySetAsSeries(time,false);
ArraySetAsSeries(open,false);
ArraySetAsSeries(high,false);
ArraySetAsSeries(low,false);
ArraySetAsSeries(close,false);
ArraySetAsSeries(tick_volume,false);
//--- цикл расчета значений индикатора
for(int i=start;i<rates_total;i++)
{
    //--- закрашиваем свечи, начиная с начальной даты
    if(ExtStartTime<=time[i])
    {
        //--- если значение не меньше правой границы, то закрашиваем свечу
        if(tick_volume[i]>=ExtRightBorder)
        {
            //--- получим данные для рисования свечи
            ExtOpenBuff[i]=open[i];
            ExtHighBuff[i]=high[i];
            ExtLowBuff[i]=low[i];
            ExtCloseBuff[i]=close[i];
            //--- цвет DodgerBlue
            ExtColorBuff[i]=0;
            //--- продолжаем цикл
            continue;
        }
        //--- если значение не больше левой границы, то закрашиваем свечу
        if(tick_volume[i]<=ExtLeftBorder)
        {
            //--- получим данные для рисования свечи
            ExtOpenBuff[i]=open[i];
            ExtHighBuff[i]=high[i];
            ExtLowBuff[i]=low[i];
            ExtCloseBuff[i]=close[i];
            //--- цвет Orange
            ExtColorBuff[i]=1;
            //--- продолжаем цикл
            continue;
        }
    }
}
}

```

```
//--- для баров, не попавших в расчет, ставим пустое значение
ExtOpenBuff[i]=INDICATOR_EMPTY_VALUE;
ExtHighBuff[i]=INDICATOR_EMPTY_VALUE;
ExtLowBuff[i]=INDICATOR_EMPTY_VALUE;
ExtCloseBuff[i]=INDICATOR_EMPTY_VALUE;
}

//--- return value of prev_calculated for next call
return(rates_total);
}
```

Смотри также

[ArrayBsearch](#)

## ArraySwap

Обменивает между собой содержимое двух динамических массивов одного типа. Для многомерных массивов количество элементов во всех измерениях кроме первого должно совпадать.

```
bool ArraySwap(
    void& array1[],      // первый массив
    void& array2[]       // второй массив
);
```

### Параметры

*array1[]*  
[in][out] Массив числового типа.

*array2[]*  
[in][out] Массив числового типа.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. В этом случае [GetLastError\(\)](#) вернет код ошибки [ERR\\_INVALID\\_ARRAY](#).

### Примечание

Функция принимает динамические массивы одинакового типа и одинаковых размерностей, кроме первой. Для целых типов знак игнорируется, т.е. [char==uchar](#)

### Пример:

```
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- массивы для хранения котировок
    double source_array[][8];
    double dest_array[][8];
    MqlRates rates[];

//--- получаем данные 20 последних свечей на текущем таймфрейме
    int copied=CopyRates(NULL,0,0,20,rates);
    if(copied<=0)
    {
        PrintFormat("CopyRates(%s,0,0,20,rates) failed, error=%d",
                   Symbol(), GetLastError());
        return;
    }
//--- зададим размер массива под количество скопированных данных
    ArrayResize(source_array,copied);
//--- заполним массив rate_array_1[] данными из rates[]
    for(int i=0;i<copied;i++)
```

```
{  
    source_array[i][0]=(double)rates[i].time;  
    source_array[i][1]=rates[i].open;  
    source_array[i][2]=rates[i].high;  
    source_array[i][3]=rates[i].low;  
    source_array[i][4]=rates[i].close;  
    source_array[i][5]=(double)rates[i].tick_volume;  
    source_array[i][6]=(double)rates[i].spread;  
    source_array[i][7]=(double)rates[i].real_volume;  
}  
//--- произведем обмен данными между source_array[] и dest_array[]  
if(!ArraySwap(source_array,dest_array))  
{  
    PrintFormat("ArraySwap(source_array,rate_array_2) failed, error code=%d",GetLastError());  
    return;  
}  
//--- убедимся, что после обмена размер массива-источника стал нулевым  
PrintFormat("ArraySwap() done: ArraySize(source_array)=%d",ArraySize(source_array))  
//--- выведем данные массива-приемника dest_array[]  
ArrayPrint(dest_array);  
}
```

#### Смотри также

[ArrayCopy](#), [ArrayFill](#), [ArrayRange](#), [ArrayIsDynamic](#)

## Преобразование 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.

Функция	Действие
<a href="#">CharToString</a>	Преобразование кода символа в односимвольную строку
<a href="#">DoubleToString</a>	Преобразование числового значения в текстовую строку с указанной точностью
<a href="#">EnumToString</a>	Преобразование значения перечисления любого типа в строку
<a href="#">NormalizeDouble</a>	Округление числа с плавающей точкой до указанной точности
<a href="#">StringtoDouble</a>	Преобразование строки, содержащей символьное представление числа, в число типа double
<a href="#">StringToInteger</a>	Преобразование строки, содержащей символьное представление числа, в число типа int
<a href="#">StringToTime</a>	Преобразование строки, содержащей время и/или дату в формате "yyyy.mm.dd [hh:mi]", в число типа datetime
<a href="#">TimeToString</a>	Преобразование значения, содержащего время в секундах, прошедшее с 01.01.1970, в строку формата "yyyy.mm.dd hh:mi"
<a href="#">IntegerToString</a>	Преобразование значения целого типа в строку указанной длины
<a href="#">ShortToString</a>	Преобразование код символа (unicode) в односимвольную строку
<a href="#">ShortArrayToString</a>	Копирует часть массива в строку
<a href="#">StringToShortArray</a>	Посимвольно копирует строку в указанное место массива типа ushort
<a href="#">CharArrayToString</a>	Преобразует код символа (ansi) в односимвольную строку
<a href="#">StringToCharArray</a>	Посимвольно копирует преобразованную из Unicode в ANSI строку в указанное место массива типа uchar

<a href="#">CharArrayToStruct</a>	Копирует <a href="#">POD-структуру</a> в массив типа uchar
<a href="#">StructToCharArray</a>	Копирует <a href="#">POD-строку</a> в массив типа uchar
<a href="#">ColorToARGB</a>	Преобразует тип color в тип uint для получения ARGB-представления цвета.
<a href="#">ColorToString</a>	Преобразует значение цвета в строку вида "R,G,B"
<a href="#">StringToColor</a>	Преобразует строку типа "R,G,B" или строку, содержащую наименование цвета, в значение типа color
<a href="#">StringFormat</a>	Преобразует число в строку в соответствие с заданным форматом

Смотри также

[Использование кодовой страницы](#)

## CharToString

Преобразование кода символа в односимвольную строку.

```
string CharToString(  
    uchar char_code           // числовой код символа  
);
```

### Параметры

*char\_code*  
[in] Код символа ANSI.

### Возвращаемое значение

Строка, содержащая символ ANSI.

### Смотри также

[StringToCharArray](#), [ShortToString](#), [StringGetCharacter](#)

## CharArrayToString

Копирует и преобразует часть массива типа uchar в возвращаемую строку.

```
string CharArrayToString(
    uchar array[],           // массив
    int start=0,             // начальная позиция в массиве
    int count=-1,            // количество символов
    uint codepage=CP_ACP     // кодовая страница
);
```

### Параметры

*array[]*

[in] Массив типа uchar.

*start=0*

[in] Позиция, с которой начинается копирование. По умолчанию 0.

*count=-1*

[in] Количество элементов массива для копирования. Определяет длину результатной строки.

По умолчанию -1, что означает копирование до конца массива, либо до встречи терминального 0.

*codepage=CP\_ACP*

[in] Значение кодовой страницы. Для наиболее употребимых [кодовых страниц](#) предусмотрены соответствующие константы.

### Возвращаемое значение

Строка.

### Смотри также

[StringToCharArray](#), [ShortArrayToString](#), [Использование кодовой страницы](#)

## CharArrayToStruct

Копирует массив типа uchar в [POD-структуру](#).

```
bool CharArrayToStruct(
    void& struct_object, // структура
    const uchar& char_array[], // массив
    uint start_pos=0 // начальная позиция в массиве
);
```

### Параметры

*struct\_object*

[in] Ссылка на любой тип [POD-структуры](#) (структурь, содержащей только простые типы данных).

*char\_array[]*

[in] Массив типа [uchar](#).

*start\_pos=0*

[in] Позиция в массиве, начиная с которой будут скопированы данные.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Смотри также

[StringToCharArray](#), [ShortArrayToString](#), [StructToCharArray](#), [Использование кодовой страницы](#),  
[FileReadStruct](#), [Объединение \(union\)](#), [MathSwap](#)

## StructToCharArray

Копирует [POD-структуру](#) в массив типа uchar.

```
bool StructToCharArray(
    const void& struct_object,           // структура
    uchar&      char_array[],           // массив
    uint        start_pos=0             // начальная позиция в массиве
);
```

### Параметры

*struct\_object*

[in] Ссылка на любой тип [POD-структуры](#) (структуры, содержащей только простые типы данных).

*char\_array[]*

[in] Массив типа [uchar](#).

*start\_pos=0*

[in] Позиция в массиве, начиная с которой будут добавлены скопированные данные.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

При копировании динамический массив автоматически расширяется ([ArrayResize](#)), если в нем недостаточно места. Если расширить массива до необходимой величины не удалось, функция вернет ошибку.

### Смотри также

[StringToCharArray](#), [ShortArrayToString](#), [CharArrayToStruct](#), [Использование кодовой страницы](#), [FileWriteStruct](#), [Объединение \(union\)](#), [MathSwap](#)

## ColorToARGB

Преобразует тип [color](#) в тип [uint](#) для получения ARGB-представления цвета. ARGB формат цвета используется при создании [графического ресурса](#), [вывода текста](#) и в классе стандартной библиотеки CCanvas.

```
uint ColorToARGB(
    color clr,           // преобразуемый цвет в формате color
    uchar alpha=255       // альфа-канал, управляющий прозрачностью цвета
);
```

### Параметры

*clr*

[in] Значение цвета в переменной типа color.

*alpha*

[in] Значение альфа-канала, для получения цвета в формате [ARGB](#). Задается значением от 0 (цвет накладываемого пикселя совсем не меняет отображения нижележащего пикселя) до 255 (цвет накладывается полностью и перекрывает собою цвет нижележащего пикселя). Прозрачность цвета в процентном выражении вычисляется как  $(1-\alpha/255)*100\%$ , то есть чем меньше значение альфа-канала, тем более прозрачен цвет.

### Возвращаемое значение

Представление цвета в формате ARGB, где в четырех байтах типа uint прописаны по порядку значения Alfa, Red, Green, Blue (альфа-канал, красный, зеленый, голубой).

### Примечание

Основным общеупотребимым форматом для описания цвета пикселя на экране дисплея в компьютерной графике является RGB, где по названиям базовых цветов задаются красная (Red), зеленая (Green) и синяя (Blue) компоненты цвета. Каждая компонента описывается одним байтом, задающим насыщенность данного цвета в интервале от 0 до 255 (от 0x00 до 0xFF в шестнадцатеричном формате). Так как белый цвет содержит в себе все цвета, то он описывается как 0xFFFFFFFF, то есть каждая из трех компонент представлена максимальным значением 0xFF.

Но в ряде задач требуется указание прозрачности цвета, чтобы описать, каким образом будет выглядеть изображение, если на него наложен цвет с некоторой долей прозрачности. Для таких случаев вводится понятие альфа-канала (Alpha), который вводится как дополнительная компонента к формату RGB. Схема формата ARGB показана на рисунке.

8	8	8	8
Alpha	Red	Green	Blue
31	30	29	28
27	26	25	24
23	22	21	20
19	18	17	16
15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	0

Значения в виде ARGB обычно указываются в шестнадцатеричном формате, где каждая пара цифр представляет по порядку значения каналов Alpha, Red, Green и Blue. Например, ARGB-цвет 80FFFFFF означает желтый цвет с непрозрачностью 50.2 %. В начале идет 0x80, которое задает 50.2% значение альфа-канала, так как это 50.2% от значения 0xFF; далее первая пара FF представляет максимальное значение красной компоненты; следующая пара FF задает такую же

силу зеленой компоненты; и последняя пара 00 представляет минимальное значение синей компоненты (отсутствие синего). Сложение зеленого и красного цвета дает желтый. Если альфа-канал не используется, запись может быть сокращена до 6 цифр RRGGBB, вот почему значения альфа-канала хранятся в верхних битах целочисленного типа uint.

В зависимости от контекста 16-ричные цифры могут записываться с префиксом '0x' или '#', например, 80FFFF00, или 0x80FFFF00, или #80FFFF00.

**Пример:**

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- зададим прозрачность
uchar alpha=0x55; // значение 0x55 означает 55/255=21.6 % прозрачности
//--- выведем преобразование в ARGB для цвета clrBlue
PrintFormat("0x%.8X - clrBlue",clrBlue);
PrintFormat("0x%.8X - clrBlue ARGB with alpha=0x55 (transparency 21.6%%)",ColorToARGB(clrBlue, alpha));
//--- выведем преобразование в ARGB для цвета clrGreen
PrintFormat("0x%.8X - clrGreen",clrGreen);
PrintFormat("0x%.8X - clrGreen ARGB with alpha=0x55 (transparency 21.6%%)",ColorToARGB(clrGreen, alpha));
//--- выведем преобразование в ARGB для цвета clrRed
PrintFormat("0x%.8X - clrRed",clrRed);
PrintFormat("0x%.8X - clrRed ARGB with alpha=0x55 (transparency 21.6%%)",ColorToARGB(clrRed, alpha));
}
```

**Смотри также**

[Ресурсы](#), [ResourceCreate\(\)](#), [TextOut\(\)](#), [Тип color](#), [Типы char, short, int и long](#)

## ColorToString

Преобразует значение цвета в строку вида "R,G,B".

```
string ColorToString(
    color color_value,           // значение цвета
    bool color_name             // выводить имя цвета или нет
);
```

### Параметры

*color\_value*

[in] Значение цвета в переменной типа color.

*color\_name*

[in] Признак необходимости возвращать имя цвета, в случае, если значение цвета совпадает с одной из предопределенных [цветовых констант](#).

### Возвращаемое значение

Строковое представление цвета в виде "R,G,B", где R, G и B представляют собой преобразованные в строку десятичные константы со значением от 0 до 255. Если задан параметр *color\_name=true*, то производится попытка преобразовать значение цвета к имени цвета.

### Пример:

```
string clr=ColorToString(C'0,255,0'); // зеленый цвет
Print(clr);

clr=ColorToString(C'0,255,0',true); // получить цветовую константу
Print(clr);
```

### Смотри также

[StringToColor](#), [ColorToARGB](#)

## DoubleToString

Преобразование числового значения в текстовую строку.

```
string DoubleToString(
    double value,           // число
    int    digits=8         // кол-во знаков после запятой
);
```

### Параметры

*value*

[in] Величина с плавающей точкой.

*digits*

[in] Формат точности. Если значение *digits* лежит в диапазоне от 0 до 16, то будет получено строковое представление числа с указанным количеством знаков после запятой. Если значение *digits* лежит в диапазоне от -1 до -16, то будет получено строковое представление числа в научном формате с указанным количеством знаков после запятой. Во всех остальных случаях строковое представление числа будет содержать 8 знаков после запятой.

### Возвращаемое значение

Строка, содержащая символьное представление числа в указанном формате точности.

### Пример:

```
Print("DoubleToString(120.0+M_PI) : ",DoubleToString(120.0+M_PI));
Print("DoubleToString(120.0+M_PI,16) : ",DoubleToString(120.0+M_PI,16));
Print("DoubleToString(120.0+M_PI,-16) : ",DoubleToString(120.0+M_PI,-16));
Print("DoubleToString(120.0+M_PI,-1) : ",DoubleToString(120.0+M_PI,-1));
Print("DoubleToString(120.0+M_PI,-20) : ",DoubleToString(120.0+M_PI,-20));
```

### Смотри также

[NormalizeDouble](#), [StringToDouble](#)

## EnumToString

Преобразование значения перечисления любого типа в текстовое представление.

```
string EnumToString(
    any_enum value          // значение из перечисления любого типа
);
```

### Параметры

*value*

[in] Значение перечисления любого типа.

### Возвращаемое значение

Строка, содержащая текстовое представление значения. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция может установить в переменной [LastError](#) следующие значения ошибок:

- **ERR\_INTERNAL\_ERROR** - ошибка среды выполнения
- **ERR\_NOT\_ENOUGH\_MEMORY** - недостаточно памяти для завершения операции
- **ERR\_INVALID\_PARAMETER** - невозможно разрешить имя значения перечисления

### Пример:

```
enum interval // перечисление именованных констант
{
    month=1,      // интервал в один месяц
    two_months,   // два месяца
    quarter,      // три месяца - квартал
    halfyear=6,   // полугодие
    year=12,      // год - 12 месяцев
};

//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
    //--- установим временной интервал равным месяцу
    interval period=month;
    Print(EnumToString(period)+"="+IntegerToString(period));

    //--- установим временной интервал равным кварталу (три месяца)
    period=quarter;
    Print(EnumToString(period)+"="+IntegerToString(period));

    //--- установим временной интервал равным году (12 месяцев)
    period=year;
    Print(EnumToString(period)+"="+IntegerToString(period));

    //--- проверим как выводится тип ордера
}
```

```
ENUM_ORDER_TYPE type=ORDER_TYPE_BUY;  
Print(EnumToString(type)+"="+IntegerToString(type));  
  
//--- проверим как выводится неверное значение  
type=WRONG_VALUE;  
Print(EnumToString(type)+"="+IntegerToString(type));  
  
// Результат выполнения:  
// month=1  
// quarter=3  
// year=12  
// ORDER_TYPE_BUY=0  
// ENUM_ORDER_TYPE::-1=-1
```

#### Смотри также

[Перечисления](#), [Input переменные](#)

## IntegerToString

Преобразует значение целого типа в строку указанной длины и возвращает полученную строку.

```
string IntegerToString(  
    long   number,           // число  
    int    str_len=0,         // длина строки на выходе  
    ushort fill_symbol=' ')  // заполнитель  
) ;
```

### Параметры

*number*

[in] Число для преобразования.

*str\_len=0*

[in] Длина строки. Если длина полученной строки окажется больше указанной, то строка не усекается. Если длина полученной строки окажется меньше, то полученная строка будет дополнена слева символом-заполнителем.

*fill\_symbol=' '*

[in] Символ-заполнитель. По умолчанию - пробел.

### Возвращаемое значение

Строка.

### Смотри также

[StringToInteger](#)

## ShortToString

Преобразует код символа (unicode) в односимвольную строку и возвращает полученную строку.

```
string ShortToString(  
    ushort symbol_code           // символ  
) ;
```

### Параметры

*symbol\_code*

[in] Код символа. Вместо кода символа можно использовать литеральную строку, содержащую символ, либо литеральную строку с двухбайтовым шестнадцатиричным кодом, соответствующему символу из таблицы Unicode.

### Возвращаемое значение

Строка.

### Смотри также

[StringToArray](#), [CharToString](#), [StringGetCharacter](#)

## ShortArrayToString

Копирует часть массива в возвращаемую строку.

```
string ShortArrayToString(
    ushort array[],           // массив
    int start=0,              // начальная позиция в массиве
    int count=-1              // количество символов
);
```

### Параметры

*array[]*

[in] Массив типа ushort (аналог типа wchar\_t).

*start=0*

[in] Позиция, с которой начинается копирование. По умолчанию 0.

*count=-1*

[in] Количество элементов массива для копирования. Определяет длину результатной строки. По умолчанию -1, что означает копирование до конца массива, либо до встречи терминального 0.

### Возвращаемое значение

Строка.

### Смотри также

[StringToShortArray](#), [CharArrayToString](#), [Использование кодовой страницы](#)

## TimeToString

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

```
string TimeToString(  
    datetime value, // число  
    int mode=TIME_DATE|TIME_MINUTES // формат вывода  
) ;
```

### Параметры

*value*

[in] Время в секундах от 00:00 1 января 1970.

*mode=TIME\_DATE|TIME\_MINUTES*

[in] 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 " .

### Возвращаемое значение

Строка.

### Смотри также

[StringToTime](#), [TimeToStruct](#)

## NormalizeDouble

Округление числа с плавающей точкой до указанной точности.

```
double NormalizeDouble(
    double value,           // нормализуемое число
    int    digits          // кол-во знаков после запятой
);
```

### Параметры

*value*

[in] Величина с плавающей точкой.

*digits*

[in] Формат точности, число цифр после десятичной точки (0-8).

### Возвращаемое значение

Значение типа double с заданной точностью.

### Примечание

Рассчитываемые значения StopLoss, TakeProfit, а также значения цены открытия отложенных ордеров, должны быть нормализованы с точностью, значение которой можно получить функцией [Digits\(\)](#).

Нужно иметь в виду, что нормализованное число при выводе в Журнал с помощью Print() может содержать большее количество знаков после запятой, чем вы ожидаете. Например,

```
double a=76.671;           // нормализованное число с 3 знаками после запятой
Print("Print(76.671)=",a); // выведем его как есть
Print("DoubleToString(a,8)=",DoubleToString(a,8)); // выведем с заданной точностью
```

выдаст в терминале:

DoubleToString(a,8)=76.67100000

Print(76.671)=76.67100000000001

### Пример:

```
double pi=M_PI;
Print("pi=",DoubleToString(pi,16));

double pi_3=NormalizeDouble(M_PI,3);
Print("NormalizeDouble(pi,3) = ",DoubleToString(pi_3,16))
;

double pi_8=NormalizeDouble(M_PI,8);
Print("NormalizeDouble(pi,8) = ",DoubleToString(pi_8,16));

double pi_0=NormalizeDouble(M_PI,0);
Print("NormalizeDouble(pi,0) = ",DoubleToString(pi_0,16));
/*
```

Результат:

```
pi= 3.1415926535897931
NormalizeDouble(pi,3)= 3.141999999999999
NormalizeDouble(pi,8)= 3.141592649999998
NormalizeDouble(pi,0)= 3.000000000000000
*/
```

#### Смотри также

[DoubleToString](#), [Вещественные типы \(double, float\)](#), [Приведение типов](#)

## StringToArray

Посимвольно копирует преобразованную из unicode в ansi строку в указанное место массива типа uchar. Функция возвращает количество скопированных элементов.

```
int StringToArray(
    string text_string,           // строка-источник
    uchar& array[],              // массив
    int start=0,                 // начальная позиция в массиве
    int count=-1,                // количество символов
    uint codepage=CP_ACP         // кодовая страница
);
```

### Параметры

*text\_string*

[in] Стока для копирования.

*array[]*

[out] Массив типа uchar.

*start=0*

[in] Позиция, с которой начинается копирование. По умолчанию 0.

*count=-1*

[in] Количество элементов массива для копирования. Определяет длину результатной строки. По умолчанию -1, что означает копирование до конца массива, либо до встречи терминального 0. Терминальный 0 также будет скопирован в массив-приемник, при этом размер динамического массива может быть увеличен при необходимости под размер строки. Если размер динамического массива больше длины строки, то размер массива уменьшен не будет.

*codepage=CP\_ACP*

[in] Значение кодовой страницы. Для наиболее употребимых [кодовых страниц](#) предусмотрены соответствующие константы.

### Возвращаемое значение

Количество скопированных элементов.

### Смотри также

[CharArrayToString](#), [StringToShortArray](#), [Использование кодовой страницы](#)

## StringToColor

Преобразует строку типа "R,G,B" или строку, содержащую наименование цвета, в значение типа color.

```
color StringToColor(  
    string color_string           // строковое представление цвета  
);
```

### Параметры

*color\_string*

[in] Строковое представление цвета типа "R,G,B" или название одного из предопределенных [Web-цветов](#).

### Возвращаемое значение

Значение цвета.

### Пример:

```
color str_color=StringToColor("0,127,0");  
Print(str_color);  
Print((string)str_color);  
//--- немного изменим цвет  
str_color=StringToColor("0,128,0");  
Print(str_color);  
Print((string)str_color);
```

### Смотри также

[ColorToString](#), [ColorToARGB](#)

## StringToDouble

Преобразование строки, содержащей символьное представление числа, в число типа double.

```
double StringToDouble(  
    string value // строка  
) ;
```

### Параметры

*value*

[in] Стока, содержащая символьное представление числа.

### Возвращаемое значение

Значение типа double.

### Смотри также

[NormalizeDouble](#), [Вещественные типы \(double, float\)](#), [Приведение типов](#)

## StringToInteger

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

```
long StringToInteger(  
    string value // строка  
) ;
```

### Параметры

*value*

[in] Стока, содержащая число.

### Возвращаемое значение

Значение типа long.

### Смотри также

[IntegerToString](#), [Вещественные типы \(double, float\)](#), [Приведение типов](#)

## StringToShortArray

Посимвольно копирует строку в указанное место массива типа ushort. Функция возвращает количество скопированных элементов.

```
int StringToShortArray(
    string text_string,           // строка-источник
    ushort& array[],             // массив
    int     start=0,              // начальная позиция в массиве
    int     count=-1              // количество символов
);
```

### Параметры

*text\_string*

[in] Стока для копирования.

*array[]*

[out] Массив типа [ushort](#) (аналог типа wchar\_t).

*start=0*

[in] Позиция, с которой начинается копирование. По умолчанию 0.

*count=-1*

[in] Количество элементов массива для копирования. Определяет длину результатной строки.

По умолчанию -1, что означает копирование до конца массива, либо до встречи терминального 0. Терминальный 0 также будет скопирован в массив-приемник, при этом размер динамического массива может быть увеличен при необходимости под размер строки. Если размер динамического массива больше длины строки, то размер массива уменьшен не будет.

### Возвращаемое значение

Количество скопированных элементов.

### Смотри также

[ShortArrayToString](#), [StringToCharArray](#), [Использование кодовой страницы](#)

## StringToTime

Преобразование строки, содержащей время и/или дату в формате "yyyy.mm.dd [hh:mi]", в число типа `datetime`.

```
datetime StringToTime(  
    const string  time_string      // строка-дата  
);
```

### Параметры

`time_string`

[in] Стока в одном из указанных форматов:

- "yyyy.mm.dd [hh:mi]"
- "yyyy.mm.dd [hh:mi:ss]"
- "yyyymmdd [hh:mi:ss]"
- "yyymmdd [hhmiss]"
- "yyyy/mm/dd [hh:mi:ss]"
- "yyyy-mm-dd [hh:mi:ss]"

### Возвращаемое значение

Значение типа [datetime](#), содержащее количество секунд, прошедших с 01.01.1970.

### Примечание

Любая последовательность символов пробела и табуляции между датой и временем считается как один пробел, чтобы не было необходимости в дополнительной обработки строки `time_string` перед вызовом `StringToTime()`.

### Смотри также

[TimeToString](#), [TimeToStruct](#)

## StringFormat

Форматирует полученные параметры и возвращает строку.

```
string StringFormat(
    string format,           // строка с описанием формата
    ...                      // параметры
);
```

### Параметры

*format*

[in] Стока, содержащая способ форматирования. Правила форматирования такие же, как и для функции [PrintFormat](#)

...

[in] Параметры, разделенные запятой.

### Возвращаемое значение

Строка.

### Пример:

```

void OnStart()
{
//--- строковые переменные
    string output_string;
    string temp_string;
    string format_string;
//--- подготовим заголовок спецификации
    temp_string=StringFormat("Спецификация контракта для %s:\n", _Symbol);
    StringAdd(output_string,temp_string);
//--- вывод значения int
    int digits=(int)SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
    temp_string=StringFormat("    SYMBOL_DIGITS = %d (количество знаков после запятой)\n",
                           digits);
    StringAdd(output_string,temp_string);
//--- вывод значения double с переменным количеством цифр после десятичной точки
    double point_value=SymbolInfoDouble(_Symbol,SYMBOL_POINT);
    format_string=StringFormat("    SYMBOL_POINT = %%.%df (значение одного пункта)\n",
                           digits);
    temp_string=StringFormat(format_string,point_value);
    StringAdd(output_string,temp_string);
//--- вывод значения int
    int spread=(int)SymbolInfoInteger(_Symbol,SYMBOL_SPREAD);
    temp_string=StringFormat("    SYMBOL_SPREAD = %d (текущий спред в пунктах)\n",
                           spread);
    StringAdd(output_string,temp_string);
//--- вывод значения int
    int min_stop=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_STOPS_LEVEL);
    temp_string=StringFormat("    SYMBOL_TRADE_STOPS_LEVEL = %d (минимальный отступ в пу-
                           min_stop);
    StringAdd(output_string,temp_string);
//--- вывод значения double без дробной части
    double contract_size=SymbolInfoDouble(_Symbol,SYMBOL_TRADE_CONTRACT_SIZE);
    temp_string=StringFormat("    SYMBOL_TRADE_CONTRACT_SIZE = %.f (размер контракта)\n",
                           contract_size);
    StringAdd(output_string,temp_string);
//--- вывод значения double с точностью по умолчанию
    double tick_size=SymbolInfoDouble(_Symbol,SYMBOL_TRADE_TICK_SIZE);
    temp_string=StringFormat("    SYMBOL_TRADE_TICK_SIZE = %f (минимальное изменение цен-
                           tick_size);
    StringAdd(output_string,temp_string);
//--- определение способа начисления свопов
    int swap_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_SWAP_MODE);
    string str_swap_mode;
    switch(swap_mode)
    {
        case SYMBOL_SWAP_MODE_DISABLED: str_swap_mode="SYMBOL_SWAP_MODE_DISABLED (нет свопов)";
        case SYMBOL_SWAP_MODE_POINTS: str_swap_mode="SYMBOL_SWAP_MODE_POINTS (в пунктах)";
        case SYMBOL_SWAP_MODE_CURRENCY_SYMBOL: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY_SYMBOL";
        case SYMBOL_SWAP_MODE_CURRENCY_MARGIN: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY_MARGIN";
        case SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT: str_swap_mode="SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT";
        case SYMBOL_SWAP_MODE_INTEREST_CURRENT: str_swap_mode="SYMBOL_SWAP_MODE_INTEREST_CURRENT";
        case SYMBOL_SWAP_MODE_INTEREST_OPEN: str_swap_mode="SYMBOL_SWAP_MODE_INTEREST_OPEN";
        case SYMBOL_SWAP_MODE_REOPEN_CURRENT: str_swap_mode="SYMBOL_SWAP_MODE_REOPEN_CURRENT";
        case SYMBOL_SWAP_MODE_REOPEN_BID: str_swap_mode="SYMBOL_SWAP_MODE_REOPEN_BID (пункты)";
    }
//--- вывод значения string
    temp_string=StringFormat("    SYMBOL_SWAP_MODE = %s\n",
                           str_swap_mode);
    StringAdd(output_string,temp_string);
//--- вывод значения double с точностью по умолчанию
    double swap_long=SymbolInfoDouble(_Symbol,SYMBOL_SWAP_LONG);
}

```

```

temp_string=StringFormat("    SYMBOL_SWAP_LONG = %f (своп на покупку)\n",
                        swap_long);
StringAdd(output_string,temp_string);
//--- вывод значения double с точностью по умолчанию
double swap_short=SymbolInfoDouble(_Symbol,SYMBOL_SWAP_SHORT);
temp_string=StringFormat("    SYMBOL_SWAP_SHORT = %f (своп на продажу)\n",
                        swap_short);
StringAdd(output_string,temp_string);
//--- определение режима торговли
int trade_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_MODE);
string str_trade_mode;
switch(trade_mode)
{
    case SYMBOL_TRADE_MODE_DISABLED: str_trade_mode="SYMBOL_TRADE_MODE_DISABLED (торговля запрещена)";
    case SYMBOL_TRADE_MODE_LONGONLY: str_trade_mode="SYMBOL_TRADE_MODE_LONGONLY (разрешены только длинные позиции)";
    case SYMBOL_TRADE_MODE_SHORTONLY: str_trade_mode="SYMBOL_TRADE_MODE_SHORTONLY (разрешены только короткие позиции)";
    case SYMBOL_TRADE_MODE_CLOSEONLY: str_trade_mode="SYMBOL_TRADE_MODE_CLOSEONLY (разрешены только закрытие открытых позиций)";
    case SYMBOL_TRADE_MODE_FULL: str_trade_mode="SYMBOL_TRADE_MODE_FULL (нет ограничений на типы позиций)";
}
//--- вывод значения string
temp_string=StringFormat("    SYMBOL_TRADE_MODE = %s\n",
                        str_trade_mode);
StringAdd(output_string,temp_string);
//--- вывод значения double в компактном виде
double volume_min=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
temp_string=StringFormat("    SYMBOL_VOLUME_MIN = %g (минимальный объем сделки)\n",
                        volume_min);
StringAdd(output_string,temp_string);
//--- вывод значения double в компактном виде
double volume_step=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_STEP);
temp_string=StringFormat("    SYMBOL_VOLUME_STEP = %g (минимальный шаг изменения объема сделки)\n",
                        volume_step);
StringAdd(output_string,temp_string);
//--- вывод значения double в компактном виде
double volume_max=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MAX);
temp_string=StringFormat("    SYMBOL_VOLUME_MAX = %g (максимальный объем сделки)\n",
                        volume_max);
StringAdd(output_string,temp_string);
//--- определение способа вычисления величины залоговых средств
int calc_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_CALC_MODE);
string str_calc_mode;
switch(calc_mode)
{
    case SYMBOL_CALC_MODE_FOREX:str_calc_mode="SYMBOL_CALC_MODE_FOREX (Forex)";break;
    case SYMBOL_CALC_MODE_FUTURES:str_calc_mode="SYMBOL_CALC_MODE_FUTURES (фьючерсы)";break;
    case SYMBOL_CALC_MODE_CFD:str_calc_mode="SYMBOL_CALC_MODE_CFD (CFD)";break;
    case SYMBOL_CALC_MODE_CFDINDEX:str_calc_mode="SYMBOL_CALC_MODE_CFDINDEX (CFD на индекс)";break;
    case SYMBOL_CALC_MODE_CFDLEVERAGE:str_calc_mode="SYMBOL_CALC_MODE_CFDLEVERAGE (CFD на кредит)";break;
    case SYMBOL_CALC_MODE_EXCH_STOCKS:str_calc_mode="SYMBOL_CALC_MODE_EXCH_STOCKS (валютные биржевые акции)";break;
    case SYMBOL_CALC_MODE_EXCH_FUTURES:str_calc_mode="SYMBOL_CALC_MODE_EXCH_FUTURES (валютные биржевые фьючерсы)";break;
    case SYMBOL_CALC_MODE_EXCH_FORTS:str_calc_mode="SYMBOL_CALC_MODE_EXCH_FORTS (валютные форвардные контракты)";break;
}
//--- вывод значения string
temp_string=StringFormat("    SYMBOL_TRADE_CALC_MODE = %s\n",
                        str_calc_mode);
StringAdd(output_string,temp_string);
//--- вывод значения double с 2 цифрами после десятичной точки
double margin_initial=SymbolInfoDouble(_Symbol,SYMBOL_MARGIN_INITIAL);
temp_string=StringFormat("    SYMBOL_MARGIN_INITIAL = %.2f (начальная маржа)\n",
                        margin_initial);
StringAdd(output_string,temp_string);
//--- вывод значения double с 2 цифрами после десятичной точки
double margin_maintenance=SymbolInfoDouble(_Symbol,SYMBOL_MARGIN_MAINTENANCE);
temp_string=StringFormat("    SYMBOL_MARGIN_MAINTENANCE = %.2f (поддерживающая маржа)\n",
                        margin_maintenance);

```

```
margin_maintenance);
StringAdd(output_string,temp_string);
//--- вывод значения int
int freeze_level=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_FREEZE_LEVEL);
temp_string=StringFormat("    SYMBOL_TRADE_FREEZE_LEVEL = %d (дистанция заморозки огнища  
freeze_level);
StringAdd(output_string,temp_string);
Print(output_string);
Comment(output_string);
/* результат выполнения
Спецификация контракта для EURUSD:
SYMBOL_DIGITS = 5 (количество знаков после запятой)
SYMBOL_POINT = 0.00001 (значение одного пункта)
SYMBOL_SPREAD = 10 (текущий спред в пунктах)
SYMBOL_TRADE_STOPS_LEVEL = 18 (минимальный отступ в пунктах для стоп-ордеров)
SYMBOL_TRADE_CONTRACT_SIZE = 100000 (размер контракта)
SYMBOL_TRADE_TICK_SIZE = 0.000010 (минимальное изменение цены)
SYMBOL_SWAP_MODE = SYMBOL_SWAP_MODE_POINTS (в пунктах)
SYMBOL_SWAP_LONG = -0.700000 (своп на покупку)
SYMBOL_SWAP_SHORT = -1.000000 (своп на продажу)
SYMBOL_TRADE_MODE = SYMBOL_TRADE_MODE_FULL (нет ограничений на торговые операции)
SYMBOL_VOLUME_MIN = 0.01 (минимальный объем сделки)
SYMBOL_VOLUME_STEP = 0.01 (минимальный шаг изменения объема сделки)
SYMBOL_VOLUME_MAX = 500 (максимальный объем сделки)
SYMBOL_TRADE_CALC_MODE = SYMBOL_CALC_MODE_FOREX (Forex)
SYMBOL_MARGIN_INITIAL = 0.00 (начальная маржа)
SYMBOL_MARGIN_MAINTENANCE = 0.00 (поддерживающая маржа)
SYMBOL_TRADE_FREEZE_LEVEL = 0 (дистанция заморозки операций в пунктах)
*/
}
```

#### Смотри также

[PrintFormat](#), [DoubleToString](#), [ColorToString](#), [TimeToString](#)

## Математические функции

Набор математических и тригонометрических функций.

Функция	Действие
<a href="#">MathAbs</a>	Возвращает абсолютное значение (значение по модулю) переданного ей числа
<a href="#">MathArccos</a>	Возвращает значение арккосинуса $x$ в радианах
<a href="#">MathArcsin</a>	Возвращает значение арксинуса $x$ в радианах
<a href="#">MathArctan</a>	Возвращает арктангенс $x$ в радианах
<a href="#">MathCeil</a>	Возвращает ближайшее сверху целое числовое значение
<a href="#">MathCos</a>	Возвращает косинус числа
<a href="#">MathExp</a>	Возвращает экспоненту числа
<a href="#">MathFloor</a>	Возвращает ближайшее снизу целое числовое значение
<a href="#">MathLog</a>	Возвращает натуральный логарифм
<a href="#">MathLog10</a>	Возвращает логарифм числа по основанию 10
<a href="#">MathMax</a>	Возвращает максимальное из двух числовых значений
<a href="#">MathMin</a>	Возвращает минимальное из двух числовых значений
<a href="#">MathMod</a>	Возвращает вещественный остаток от деления двух чисел
<a href="#">MathPow</a>	Возводит основание в указанную степень
<a href="#">MathRand</a>	Возвращает псевдослучайное целое число в диапазоне от 0 до 32767
<a href="#">MathRound</a>	Округляет число до ближайшего целого
<a href="#">MathSin</a>	Возвращает синус числа
<a href="#">MathSqrt</a>	Возвращает квадратный корень
<a href="#">MathSrand</a>	Устанавливает начальное состояние генератора псевдослучайных целых чисел
<a href="#">MathTan</a>	Возвращает тангенс числа
<a href="#">MathIsValidNumber</a>	Проверяет корректность действительного числа

<a href="#"><u>MathExprm1</u></a>	Возвращает значение выражения <code>MathExp(x)-1</code>
<a href="#"><u>MathLog1p</u></a>	Возвращает значение выражения <code>MathLog(1+x)</code>
<a href="#"><u>MathArccosh</u></a>	Возвращает значение гиперболического арккосинуса
<a href="#"><u>MathArcsinh</u></a>	Возвращает значение гиперболического арксинуса
<a href="#"><u>MathArctanh</u></a>	Возвращает значение гиперболического арктангенса
<a href="#"><u>MathCosh</u></a>	Возвращает гиперболический косинус
<a href="#"><u>MathSinh</u></a>	Возвращает гиперболический синус
<a href="#"><u>MathTanh</u></a>	Возвращает гиперболический тангенс
<a href="#"><u>MathSwap</u></a>	Меняет порядок байтов в значении типа <code>ushort/uint/ulong</code>

## MathAbs

Возвращает абсолютное значение (значение по модулю) переданного ей числа.

```
double MathAbs(  
    double value           // число  
) ;
```

### Параметры

*value*

[in] Числовая величина.

### Возвращаемое значение

Значение типа double, больше или равно нулю.

### Примечание

Вместо функции MathAbs() можно использовать функцию [fabs\(\)](#).

## MathArccos

Возвращает значение арккосинуса  $x$  в диапазоне 0 к  $\pi$  в радианах.

```
double MathArccos(  
    double value // -1<val<1  
) ;
```

### Параметры

*value*

[in] Значение *value* между -1 и 1, арккосинус которого должен быть вычислен.

### Возвращаемое значение

Арккосинус числа в радианах. Если *value* меньше -1 или больше 1, функция возвращает NaN (неопределенное значение).

### Примечание

Вместо функции MathArccos() можно использовать функцию [acos\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathArcsin

Возвращает арксинус  $x$  в диапазоне от  $-\pi/2$  до  $\pi/2$  радианов.

```
double MathArcsin(  
    double value          // -1<value<1  
) ;
```

### Параметры

*value*

[in] Значение *value* между -1 и 1, арксинус которого должен быть вычислен.

### Возвращаемое значение

Арксинус числа *value* в радианах в диапазоне от  $-\pi/2$  до  $\pi/2$  радианов. Если *value* меньше -1 или больше 1, функция возвращает NaN (неопределенное значение).

### Примечание

Вместо функции MathArcsin() можно использовать функцию [asin\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathArctan

Возвращает арктангенс x. Если x равен 0, функция возвращает 0.

```
double MathArctan(  
    double value           // тангенс  
) ;
```

### Параметры

*value*

[in] Число, представляющее тангенс.

### Возвращаемое значение

MathArctan возвращает значение в диапазоне от  $-\pi/2$  до  $\pi/2$  радианов.

### Примечание

Вместо функции MathArctan() можно использовать функцию [atan\(\)](#).

## MathCeil

Возвращает ближайшее сверху целое числовое значение.

```
double MathCeil(  
    double val        // число  
);
```

### Параметры

*val*

[in] Числовая величина.

### Возвращаемое значение

Числовое значение, представляющую наименьшее целое число, которое больше или равно *val*.

### Примечание

Вместо функции MathCeil() можно использовать функцию [ceil\(\)](#).

## MathCos

Функция возвращает косинус угла.

```
double MathCos(  
    double value           // число  
) ;
```

### Параметры

*value*

[in] Угол в радианах.

### Возвращаемое значение

Значение типа double в диапазоне от -1 до 1.

### Примечание

Вместо функции MathCos() можно использовать функцию [cos\(\)](#).

## MathExp

Возвращает значение числа е в степени d.

```
double MathExp(  
    double value           // степень для числа е  
);
```

### Параметры

*value*

[in] Число, определяющее степень.

### Возвращаемое значение

Число типа double. При переполнении функция возвращает INF (бесконечность), в случае потери порядка точности MathExp возвращает 0.

### Примечание

Вместо функции MathExp() можно использовать функцию [exp\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathFloor

Возвращает ближайшее снизу целое числовое значение.

```
double MathFloor(  
    double value // число  
) ;
```

### Параметры

*val*

[in] Числовое значение.

### Возвращаемое значение

Числовое значение, представляющее наибольшее целое число, которое меньше или равно *value*.

### Примечание

Вместо функции *MathFloor()* можно использовать функцию [floor\(\)](#).

## MathLog

Возвращает натуральный логарифм.

```
double MathLog(  
    double value           // число для взятия логарифма  
);
```

### Параметры

*val*

[in] Значение, логарифм которого должен быть вычислен.

### Возвращаемое значение

Натуральный логарифм *value* в случае успеха. Если значение *val* отрицательно, функция возвращает NaN (неопределенное значение). Если *value* равно 0, функция возвращает INF (бесконечность).

### Примечание

Вместо функции MathLog() можно использовать функцию [log\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathLog

Возвращает логарифм числа по основанию 10.

```
double MathLog10(  
    double val // число для взятия логарифма  
);
```

### Параметры

*val*

[in] Значение, десятичный логарифм которого должен быть вычислен.

### Возвращаемое значение

Десятичный логарифм *val* в случае успеха. Если значение *val* отрицательно, функция возвращает NaN (неопределенное значение). Если *val* равно 0, функция возвращает INF (бесконечность).

### Примечание

Вместо функции MathLog10() можно использовать функцию [log10\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathMax

Функция возвращает максимальное из двух числовых значений.

```
double MathMax(
    double value1,      // первое число
    double value2       // второе число
);
```

### Параметры

*value1*

[in] Первое числовое значение.

*value2*

[in] Второе числовое значение.

### Возвращаемое значение

Большее из двух чисел.

### Примечание

Вместо функции MathMax() можно использовать функцию [fmax\(\)](#). Функции [fmax\(\)](#), [fmin\(\)](#), [MathMax\(\)](#), [MathMin\(\)](#) могут работать с целыми типами без преобразования к типу double.

Если в функцию передаются параметры разных типов, то параметр младшего типа автоматически [приводится](#) к старшему типу. Тип возвращаемого значения соответствует старшему типу.

Если передаются данные одного типа, то никакого преобразования не производится.

## MathMin

Функция возвращает минимальное из двух числовых значений.

```
double MathMin(
    double value1,      // первое число
    double value2       // второе число
);
```

### Параметры

*value1*

[in] Первое числовое значение.

*value2*

[in] Второе числовое значение.

### Возвращаемое значение

Меньшее из двух чисел.

### Примечание

Вместо функции MathMin() можно использовать функцию [fmin\(\)](#). Функции [fmax\(\)](#), [fmin\(\)](#), [MathMax\(\)](#), [MathMin\(\)](#) могут работать с целыми типами без преобразования к типу double.

Если в функцию передаются параметры разных типов, то параметр младшего типа автоматически [приводится](#) к старшему типу. Тип возвращаемого значения соответствует старшему типу.

Если передаются данные одного типа, то никакого преобразования не производится.

## MathMod

Возвращает вещественный остаток от деления двух чисел.

```
double MathMod(  
    double value,           // делимое  
    double value2          // делитель  
);
```

### Параметры

*value*

[in] Значение делимого.

*value2*

[in] Значение делителя.

### Возвращаемое значение

Функция MathMod рассчитывает вещественный остаток  $f$  от  $val / y$  таким образом, что  $val = i * y + f$ , где  $i$  является целым числом,  $f$  имеет тот же знак, что и  $val$ , и абсолютное значение  $f$  меньше, чем абсолютное значение  $y$ .

### Примечание

Вместо функции MathMod() можно использовать функцию [fmod\(\)](#).

## MathPow

Возводит основание в указанную степень.

```
double MathPow(  
    double base,           // основание  
    double exponent        // показатель степени  
);
```

### Параметры

*base*

[in] Основание.

*exponent*

[in] Значение степени.

### Возвращаемое значение

Значение основания, возведенного в указанную степень.

### Примечание

Вместо функции MathPow() можно использовать функцию [pow\(\)](#).

## MathRand

Возвращает псевдослучайное целое число в диапазоне от 0 до 32767.

```
int MathRand();
```

### Возвращаемое значение

Целое число в диапазоне от 0 до 32767.

### Примечание

Перед первым вызовом функции необходимо использовать функцию [MathSrand](#), чтобы перевести генератор псевдослучайных чисел в начальное состояние.

### Примечание

Вместо функции MathRand() можно использовать функцию [rand\(\)](#).

## MathRound

Возвращает значение, округленное до ближайшего целого числа указанного числового значения.

```
double MathRound(  
    double value           // округляемое значение  
);
```

### Параметры

*value*

[in] Числовая величина для округления.

### Возвращаемое значение

Значение, округленное до ближайшего целого числа.

### Примечание

Вместо функции MathRound() можно использовать функцию [round\(\)](#).

## MathSin

Возвращает синус указанного угла.

```
double MathSin(  
    double value           // число  
) ;
```

### Параметры

*value*

[in] Угол в радианах.

### Возвращаемое значение

Синус угла, указанного в радианах. Возвращает значение в диапазоне от -1 до 1.

### Примечание

Вместо функции MathSin() можно использовать функцию [sin\(\)](#).

## MathSqrt

Возвращает квадратный корень числа.

```
double MathSqrt(  
    double value           // положительное число  
) ;
```

### Параметры

*value*

[in] Положительная числовая величина.

### Возвращаемое значение

Квадратный корень числа *value*. Если *value* отрицательно, MathSqrt возвращает NaN (неопределенное значение).

### Примечание

Вместо функции MathSqrt() можно использовать функцию [sqrt\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathStrand

Устанавливает начальное состояние для генерации ряда псевдослучайных целых чисел.

```
void MathStrand(
    int seed      // инициализирующее число
);
```

### Параметры

*seed*

[in] Начальное число для ряда случайных чисел.

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

Функция [MathRand\(\)](#) предназначена для генерации последовательности псевдослучайных чисел. Вызов MathStrand() с определенным инициализирующими числом позволяет получать всегда одну и ту же последовательность псевдослучайных чисел.

Для гарантированного получения неповторяющейся последовательности используйте вызов MathStrand(GetTickCount()), так как значение [GetTickCount\(\)](#) увеличивается с момента запуска операционной системы и не повторяется в течение 49 дней, пока не переполнится встроенный счетчик миллисекунд. Использование MathStrand(TimeCurrent()) не подходит по той причине, что функция [TimeCurrent\(\)](#) возвращает время прихода последнего тика, которое может не меняться долгое время, например, в выходные дни.

Инициализацию генератора псевдослучайных чисел с помощью MathStrand() для индикаторов и экспертов лучше всего делать в обработчике OnInit(), это избавит от последующих многократных перезапусков генератора в OnTick() и OnCalculate().

Вместо функции MathStrand() можно использовать функцию [rand\(\)](#).

### Пример:

```
#property description "Индикатор демонстрирует центральную предельную теорему, которая гласит, что сумма достаточно большого количества слабо зависимых случайных величин, имеющих примерно одинаковые масштабы (ни одно из слагаемых не доминирует), имеет распределение, близкое к нормальному."
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- свойства графического построения
#property indicator_label1 "Label"
#property indicator_type1 DRAW_HISTOGRAM
#property indicator_color1 clrRoyalBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 5
//--- input переменная
input int sample_number=10;
```

```

//--- индикаторный буфер для отрисовки распределения
double      LabelBuffer[];
//--- счетчик тиков
double      ticks_counter;
//+-----+
//| Custom indicator initialization function           |
//+-----+
void OnInit()
{
//--- связывание массива и индикаторного буфера
    SetIndexBuffer(0,LabelBuffer,INDICATOR_DATA);
//--- развернем индикаторный буфер из настоящего в прошлое
    ArraySetAsSeries(LabelBuffer,true);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- инициализируем счетчик тиков
    ticks_counter=0;
}
//+-----+
//| Custom indicator iteration function               |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- при нулевом счетчике обнулим буфер индикатора
    if(ticks_counter==0) ArrayInitialize(LabelBuffer,0);
//--- увеличим счетчик
    ticks_counter++;
//--- нужно периодически сбрасывать счетчик тиков, чтобы оживлять распределение
    if(ticks_counter>100)
    {
        Print("Обнулили значения индикатора, начнем заполнять ячейки снова");
        ticks_counter=0;
    }
//--- получим выборку случайных значений, как сумму трех чисел от 0 до 7
    for(int i=0;i<sample_number;i++)
    {
//--- вычисление индекса ячейки, куда выпадет случайное число как сумма трех дру
        int rand_index=0;
//--- получим три случайных числа от 0 до 7
        for(int k=0;k<3;k++)

```

```
{  
    //--- остаток от деления на 7 вернет значение от 0 до 6  
    rand_index+=MathRand()%7;  
}  
//--- увеличим на единицу значение в ячейке с номером rand_index  
LabelBuffer[rand_index]++;  
}  
//--- выход из обработчика OnCalculate()  
return(rates_total);  
}
```

## MathTan

Возвращает тангенс числа.

```
double MathTan(  
    double rad          // аргумент в радианах  
);
```

### Параметры

*rad*

[in] Угол в радианах.

### Возвращаемое значение

Тангенс числа *rad*. Если *rad* больше или равен 263 или меньше или равен -263, то происходит потеря значения и функция возвращает неопределенное число.

### Примечание

Вместо функции MathTan() можно использовать функцию [tan\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathIsValidNumber

Проверяет корректность действительного числа

```
bool MathIsValidNumber(
    double number           // число для проверки
);
```

### Параметры

*number*

[in] Проверяемое число.

### Возвращаемое значение

Возвращает true, если проверяемое значение является допустимым вещественным числом. Если проверяемое значение является плюс или минус бесконечностью, либо "не числом" (NaN - not a number), функция возвращает false.

### Пример:

```
double abnormal=MathArcsin(2.0);
if(!MathIsValidNumber(abnormal)) Print("Внимание! MathArcsin(2.0) = ",abnormal);
```

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathExp1

Возвращает значение выражения MathExp(x)-1.

```
double MathExp1(  
    double value           // степень для числа e  
);
```

### Параметры

*value*

[in] Число, определяющее степень.

### Возвращаемое значение

Число типа double. При переполнении функция возвращает INF (бесконечность), в случае потери порядка точности MathExp1 возвращает 0.

### Примечание

При значениях *x* близком к 0, функция MathExp1(*x*) даёт гораздо более точные значения, чем MathExp(*x*)-1.

Вместо функции MathExp1() можно использовать функцию [exprm1\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathLog1p

Возвращает значение выражения MathLog(1+x).

```
double MathLog1p(  
    double value           // число для взятия логарифма  
) ;
```

### Параметры

*val*

[in] Значение, логарифм которого должен быть вычислен.

### Возвращаемое значение

Натуральный логарифм значения (*value*+1) в случае успеха. Если *value* < -1, то функция возвращает NaN (неопределенное значение). Если *value* равно -1, функция возвращает INF (бесконечность).

### Примечание

При значениях *x* близком к 0, функция MathLog1p(*x*) даёт гораздо более точные значения, чем MathLog(1+*x*).

Вместо функции MathLog1p() можно использовать функцию [log1p\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathArccosh

Возвращает значение гиперболического арккосинуса.

```
double MathArccosh(
    double value      // 1 <= value < ∞
);
```

### Параметры

*val*

[in] Значение *value*, гиперболический арккосинус которого должен быть вычислен.

### Возвращаемое значение

Гиперболический арккосинус числа. Если *value* меньше +1, функция возвращает NaN (неопределенное значение).

### Примечание

Вместо функции MathArccosh() можно использовать функцию [acosh\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathArcsinh

Возвращает значение гиперболического арксинуса.

```
double MathArcsinh(  
    double value // -∞ < value < +∞  
) ;
```

### Параметры

*val*

[in] Значение *value*, арксинус которого должен быть вычислен.

### Возвращаемое значение

Гиперболический арксинус числа.

### Примечание

Вместо функции MathArcsinh() можно использовать функцию [asinh\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathArctanh

Возвращает значение гиперболического арктангенса.

```
double MathArctanh(  
    double value           // значение в диапазоне -1 < value < 1  
);
```

### Параметры

*value*

[in] Число в диапазоне  $-1 < \text{value} < 1$ , представляющее тангенс.

### Возвращаемое значение

Гиперболический арктангенс числа.

### Примечание

Вместо функции MathArctanh() можно использовать функцию [atanh\(\)](#).

## MathCosh

Возвращает гиперболический косинус числа.

```
double MathCosh(  
    double value           // число  
) ;
```

### Параметры

*value*

[in] Значение.

### Возвращаемое значение

Гиперболический косинус числа, значение в диапазоне от +1 до плюс бесконечности.

### Примечание

Вместо функции MathCosh() можно использовать функцию [cosh\(\)](#).

## MathSinh

Возвращает гиперболический синус числа.

```
double MathSinh(  
    double value           // число  
) ;
```

### Параметры

*value*

[in] Значение.

### Возвращаемое значение

Гиперболический синус числа.

### Примечание

Вместо функции MathSinh() можно использовать функцию [sinh\(\)](#).

## MathTanh

Возвращает гиперболический тангенс числа.

```
double MathTanh(  
    double value           // число  
) ;
```

### Параметры

*value*

[in] Значение.

### Возвращаемое значение

Гиперболический тангенс числа, значение в диапазоне от -1 до +1.

### Примечание

Вместо функции MathTanh() можно использовать функцию [tanh\(\)](#).

### Смотри также

[Вещественные типы \(double, float\)](#)

## MathSwap

Меняет порядок байтов в значении типа [ushort](#).

```
ushort MathSwap(
    ushort value      // значение
);
```

### Параметры

*value*

[in] Значение для смены порядка байтов.

### Возвращаемое значение

Значение ushort с обратным порядком байтов.

## MathSwap

Меняет порядок байтов в значении типа [uint](#).

```
uint MathSwap(
    uint value       // значение
);
```

### Параметры

*value*

[in] Значение для смены порядка байтов.

### Возвращаемое значение

Значение uint с обратным порядком байтов.

## MathSwap

Меняет порядок байтов в значении типа [ulong](#).

```
ulong MathSwap(
    ulong value      // значение
);
```

### Параметры

*value*

[in] Значение для смены порядка байтов.

### Возвращаемое значение

Значение ulong с обратным порядком байтов.

### Смотри также

[Сетевые функции](#), [SocketRead](#), [SocketSend](#), [SocketTlsRead](#), [SocketTlsReadAvailable](#), [SocketTlsSend](#)

## Строковые функции

Группа функций, предназначенных для работы с данными типа [string](#).

Функция	Действие
<a href="#">StringAdd</a>	Присоединяет к концу строки по месту указанную подстроку
<a href="#">StringBufferLen</a>	Возвращает размер буфера, распределенного для строки
<a href="#">StringCompare</a>	Сравнивает две строки и возвращает 1, если первая строка больше второй; 0 - если строки равны; -1 (минус один) - если первая строка меньше второй
<a href="#">StringConcatenate</a>	Формирует строку из переданных параметров
<a href="#">StringFill</a>	Заполняет указанную строку по месту указанными символами
<a href="#">StringFind</a>	Поиск подстроки в строке
<a href="#">StringGetCharacter</a>	Возвращает значение символа, расположенного в указанной позиции строки
<a href="#">StringInit</a>	Инициализирует строку указанными символами и обеспечивает указанный размер строки
<a href="#">StringLen</a>	Возвращает число символов в строке
<a href="#">StringSetLength</a>	Устанавливает для строки указанную длину в символах
<a href="#">StringReplace</a>	Заменяет в строке все найденные подстроки на заданную последовательность символов.
<a href="#">StringReserve</a>	Резервирует в памяти для строки буфер указанного размера
<a href="#">StringSetCharacter</a>	Возвращает копию строки с измененным значением символа в указанной позиции
<a href="#">StringSplit</a>	Получает из указанной строки подстроки по заданному разделителю и возвращает количество полученных подстрок
<a href="#">StringSubstr</a>	Извлекает подстроку из текстовой строки, начинающейся с указанной позиции
<a href="#">StringToLower</a>	Преобразует все символы указанной строки в строчные (маленькие) по месту
<a href="#">StringToUpper</a>	Преобразует все символы указанной строки в прописные (большие) по месту

<a href="#"><u>StringTrimLeft</u></a>	Удаляет символы перевода каретки, пробелы и символы табуляции с начала строки
<a href="#"><u>StringTrimRight</u></a>	Удаляет символы перевода каретки, пробелы и символы табуляции в конце строки

## StringAdd

Присоединяет к концу строки по месту указанную подстроку.

```
bool StringAdd(
    string& string_var,           // строка, к которой добавляем
    string    add_substring      // добавляемая строка
);
```

### Параметры

*string\_var*

[in][out] Стока, которая будет дополнена.

*add\_substring*

[in] Стока, которая будет добавлена в конец исходной строки.

### Возвращаемое значение

В случае успешного выполнения возвращает true, иначе false. Для получения кода ошибки нужно вызвать функцию [GetLastError\(\)](#).

### Пример:

```
void OnStart()
{
    long length=1000000;
    string a="a",b="b",c;
//--- первый способ
    uint start=GetTickCount(),stop;
    long i;
    for(i=0;i<length;i++)
    {
        c=a+b;
    }
    stop=GetTickCount();
    Print("time for 'c = a + b' = ",(stop-start)," milliseconds, i = ",i);

//--- второй способ
    start=GetTickCount();
    for(i=0;i<length;i++)
    {
        StringAdd(a,b);
    }
    stop=GetTickCount();
    Print("time for 'StringAdd(a,b)' = ",(stop-start)," milliseconds, i = ",i);

//--- третий способ
    start=GetTickCount();
    a="a"; // заново инициализируем переменную a
    for(i=0;i<length;i++)
```

```
{  
    StringConcatenate(c,a,b);  
}  
stop=GetTickCount();  
Print("time for 'StringConcatenate(c,a,b)' = ",(stop-start)," milliseconds, i = ",i);  
}
```

Смотри также

[StringConcatenate](#), [StringSplit](#), [StringSubstr](#)

## StringBufferLen

Возвращает размер буфера, распределенного для строки.

```
int StringBufferLen(
    string string_var          // строка
)
```

### Параметры

*string\_var*

[in] Стока.

### Возвращаемое значение

Значение 0 означает, что строка - константная и содержимое буфера менять нельзя. -1 означает, что строка принадлежит клиентскому терминалу и изменение содержимого буфера может привести к неопределенным результатам.

### Пример:

```
void OnStart()
{
    long length=1000;
    string a="a",b="b";
//---
    long i;
    Print("before: StringBufferLen(a) = ",StringBufferLen(a),
          " StringLen(a) = ",StringLen(a));
    for(i=0;i<length;i++)
    {
        StringAdd(a,b);
    }
    Print("after: StringBufferLen(a) = ",StringBufferLen(a),
          " StringLen(a) = +-",StringLen(a));
}
```

### Смотри также

[StringAdd](#), [StringInit](#), [StringLen](#), [StringFill](#)

## StringCompare

Сравнивает между собой две строки и возвращает результат сравнения в виде целого числа.

```
int StringCompare(
    const string& string1,           // первая сравниваемая строка
    const string& string2,           // вторая сравниваемая строка
    bool         case_sensitive=true // режим учета регистра букв при сравнении
);
```

### Параметры

*string1*

[in] Первая строка.

*string2*

[in] Вторая строка.

*case\_sensitive=true*

[in] Режим учета регистра букв. Если значения равно true, то "A">>"a". Если значение равно false, то "A"="a". По умолчанию значение параметра равно true.

### Возвращаемое значение

- -1 (минус один), если *string1*<*string2*
- 0 (ноль), если *string1*=*string2*
- 1 (один), если *string1*>*string2*

### Примечание

Строки сравниваются посимвольно, символы сравниваются в алфавитном порядке в соответствии с текущей кодовой страницей.

### Пример:

```
void OnStart()
{
//--- что больше, яблоко или дом?
string s1="Apple";
string s2="home";

//--- сравним с учетом регистра
int result1=StringCompare(s1,s2);
if(result1>0) PrintFormat("Сравнение с учетом регистра: %s > %s",s1,s2);
else
{
    if(result1<0)PrintFormat("Сравнение с учетом регистра: %s < %s",s1,s2);
    else PrintFormat("Сравнение с учетом регистра: %s = %s",s1,s2);
}

//--- сравним без учета регистра
int result2=StringCompare(s1,s2,false);
if(result2>0) PrintFormat("Сравнение без учета регистра: %s > %s",s1,s2);
else
```

```
{  
    if(result2<0) PrintFormat("Сравнение без учета регистра: %s < %s",s1,s2);  
    else PrintFormat("Сравнение без учета регистра: %s = %s",s1,s2);  
}  
/* Результат:  
Сравнение с учетом регистра: Apple < home  
Сравнение без учета регистра: Apple < home  
*/  
}
```

#### Смотри также

[Тип string](#), [CharToString\(\)](#), [ShortToString\(\)](#), [StringToCharArray\(\)](#), [StringToShortArray\(\)](#),  
[StringGetCharacter\(\)](#), [Использование кодовой страницы](#)

## StringConcatenate

Формирует строку из переданных параметров и возвращает размер сформированной строки. Параметры могут иметь любой тип. Количество параметров не может быть меньше 2 и не может превышать 64.

```
int StringConcatenate(
    string& string_var,      // строка для формирования
    void argument1           // первый параметр любого простого типа
    void argument2           // второй параметр любого простого типа
    ...
    );                      // следующий параметр любого простого типа
```

### Параметры

*string\_var*

[out] Стока, которая будет сформирована в результате конкатенации.

*argumentN*

[in] Любые значения, разделенные запятыми. От 2 до 63 параметров любого типа.

### Возвращаемое значение

Возвращает длину строки, сформированной путем конкатенации преобразованных в тип `string` параметров. Параметры преобразуются в строки по тем же правилам, что и в функциях [Print\(\)](#) и [Comment\(\)](#).

### Смотри также

[StringAdd](#), [StringSplit](#), [StringSubstr](#)

## StringFill

Заполняет указанную строку по месту указанными символами.

```
bool StringFill(
    string& string_var,           // строка для заполнения
    ushort character             // символ, которым будет заполнена строка
);
```

### Параметры

*string\_var*  
 [in][out] Стока, которая будет заполнена указанным символом.

*character*  
 [in] Символ, которым будет заполнена строка.

### Возвращаемое значение

В случае успешного выполнения возвращает true, иначе false. Для получения кода ошибки нужно вызвать функцию [GetLastError\(\)](#).

### Примечание

Заполнение строки по месту означает, что символы вставляются непосредственно в строку без промежуточных операций создания новой строки и копирования. Это позволяет уменьшить время работы со строкой в данной функции.

### Пример:

```
void OnStart()
{
    string str;
    StringInit(str,20,'_');
    Print("str = ",str);
    StringFill(str,0);
    Print("str = ",str,": StringBufferLen(str) = ", StringBufferLen(str));
}
```

// Результат  
// str = \_\_\_\_\_  
// str = : StringBufferLen(str) = 20  
//

### Смотри также

[StringBufferLen](#), [StringLen](#), [StringInit](#)

## StringFind

Поиск подстроки в строке.

```
int StringFind(
    string string_value,           // строка, в которой ищем
    string match_substring,        // что ищем
    int     start_pos=0           // с какой позиции начинать поиск
);
```

### Параметры

*string\_value*

[in] Стока, в которой производится поиск.

*match\_substring*

[in] Искомая подстрока.

*start\_pos=0*

[in] Позиция в строке, с которой должен быть начат поиск.

### Возвращаемое значение

Возвращает номер позиции в строке, с которой начинается искомая подстрока, либо -1, если подстрока не найдена.

### Смотри также

[StringSubstr](#), [StringGetCharacter](#), [StringLen](#), [StringLen](#)

## StringGetCharacter

Возвращает значение символа, расположенного в указанной позиции строки.

```
ushort StringGetCharacter(
    string string_value,      // строка
    int     pos              // позиция символа в строке
);
```

### Параметры

*string\_value*

[in] Стока.

*pos*

[in] Позиция символа в строке. Может быть от 0 до [StringLen\(text\)](#) -1.

### Возвращаемое значение

Код символа либо 0 в случае какой-либо ошибки. Для получения кода [ошибки](#) нужно вызвать функцию [GetLastError\(\)](#).

### Смотри также

[StringSetCharacter](#), [StringBufferLen](#), [StringLen](#), [StringFill](#), [StringInit](#), [StringToCharArray](#),  
[String.ToShortArray](#)

## StringInit

Инициализирует строку указанными символами и обеспечивает указанный размер строки.

```
bool StringInit(
    string& string_var,           // строка для инициализации
    int      new_len=0,          // требуемая длина строки после инициализации
    ushort   character=0         // символ, которым будет заполнена строка
);
```

### Параметры

*string\_var*

[in][out] Стока, которая должна быть инициализирована или деинициализирована.

*new\_len=0*

[in] Длина строки после инициализации. Если размер=0, то деинициализирует строку, то есть, буфер строки освобождается и адрес буфера обнуляется.

*character=0*

[in] Символ для заполнения строки.

### Возвращаемое значение

В случае успешного выполнения возвращает true, иначе false. Для получения кода ошибки нужно вызвать функцию [GetLastError\(\)](#).

### Примечание

Если character=0 и размер new\_len>0, то будет распределен буфер строки указанного размера и заполнен нулями. Размер строки будет равен нулю, так как весь буфер заполнен терминаторами строки.

### Пример:

```
void OnStart()
{
//---
    string str;
    StringInit(str,200,0);
    Print("str = ",str,": StringBufferLen(str) = ",
          StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
}
/* Результат
str = : StringBufferLen(str) = 200  StringLen(str) = 0
*/
```

### Смотри также

[StringBufferLen](#), [StringLen](#)

## StringLen

Возвращает число символов в строке.

```
int StringLen(
    string string_value          // строка
);
```

### Параметры

*string\_value*

[in] Стока для вычисления длины.

### Возвращаемое значение

Количество символов в строке без учета завершающего нуля.

### Смотри также

[StringBufferLen](#), [StringTrimLeft](#), [StringTrimRight](#), [StringToArray](#), [StringToShortArray](#)

## StringSetLength

Устанавливает для строки указанную длину в символах.

```
bool StringSetLength(
    string& string_var,           // строка
    uint      new_length         // новая длина строки
);
```

### Параметры

*string\_var*

[in][out] Стока, для которой необходимо задать новую длину в символах.

*new\_capacity*

[in] Требуемая длина строки в символах. Если новая длина *new\_length* меньше текущего размера, то непоместившиеся символы будут отброшены.

### Возвращаемое значение

В случае успешного выполнения возвращает true, иначе false. Для получения кода ошибки нужно вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция StringSetLength() не изменяет размера буфера, отведенного под строку.

### Смотри также

[StringLen](#), [StringBufferLen](#), [StringReserve](#) [StringInit](#), [StringSetCharacter](#)

## StringReplace

Заменяет в строке все найденные подстроки на заданную последовательность символов.

```
int StringReplace(
    string& str,           // строка, в которой будет осуществляться замена
    const string find,     // искомая подстрока
    const string replacement // подстрока, которая будет вставлена в найденные
);
```

### Параметры

*str*

[in][out] Страна, в которой необходимо произвести замены.

*find*

[in] Искомая подстрока для замены.

*replacement*

[in] Подстрока, которая будет вставлена вместо найденной.

### Возвращаемое значение

Количество произведенных замен в случае успеха, в случае ошибки -1. Для получения кода ошибки нужно вызвать функцию [GetLastError\(\)](#).

### Примечание

Если функция успешно отработала, но замены не произведены (не найдена заменяемая подстрока), то возвращается 0.

Причиной ошибки могут быть неверные параметры *str* или *find* (пустая или неинициализированная строка, см. [StringInit\(\)](#)). Кроме того, ошибка возникнет, если для завершения замен недостаточно памяти.

### Пример:

```
string text="The quick brown fox jumped over the lazy dog.";
int replaced=StringReplace(text,"quick","slow");
replaced+=StringReplace(text,"brown","black");
replaced+=StringReplace(text,"fox","bear");
Print("Replaced: ", replaced,". Result=",text);

// Результат
// Replaced: 3. Result=The slow black bear jumped over the lazy dog.
//
```

### Смотри также

[StringSetCharacter\(\)](#), [StringSubstr\(\)](#)

## StringReserve

Резервирует в памяти для строки буфер указанного размера.

```
bool StringReserve(
    string& string_var,           // строка
    uint      new_capacity       // размер буфера для хранения строки
);
```

### Параметры

*string\_var*

[in][out] Стока, для которой необходимо изменить размер буфера.

*new\_capacity*

[in] Требуемый размер буфера под строку. Если новый размер *new\_capacity* меньше длины строки, то размер текущего буфера не изменится.

### Возвращаемое значение

В случае успешного выполнения возвращает true, иначе false. Для получения кода [ошибки](#) нужно вызвать функцию [GetLastError\(\)](#).

### Примечание

В общем случае размер строки не равен размеру буфера, предназначенному для хранения строки. Как правило при создании строки буфер под неё выделяется с запасом. Функция *StringReserve()* позволяет управлять размером буфера и указывает оптимальный размер для будущих операций.

В отличие от [StringInit\(\)](#) функция *StringReserve()* не изменяет содержимое строки и не заполняет её символами.

### Пример:

```
void OnStart()
{
    string s;
//--- проверим скорость работы без использования StringReserve
    ulong t0=GetMicrosecondCount();
    for(int i=0; i< 1024; i++)
        s+=" "+(string)i;
    ulong msc_no_reserve=GetMicrosecondCount()-t0;
    s=NULL;
//--- теперь замерим с использованием StringReserve
    StringReserve(s,1024 * 3);
    t0=GetMicrosecondCount();
    for(int i=0; i< 1024; i++)
        s+=" "+(string)i;
    ulong msc_reserve=GetMicrosecondCount()-t0;
//--- проверим время
    Print("Test with StringReserve passed for "+(string)msc_reserve+" msc");
    Print("Test without StringReserve passed for "+(string)msc_no_reserve+" msc");
```

```
/* Результат:  
Test with StringReserve passed for 50 msc  
Test without StringReserve passed for 121 msc  
*/  
}
```

**Смотри также**

[StringBufferLen](#), [StringSetLength](#), [StringInit](#), [StringSetCharacter](#)

## StringSetCharacter

Возвращает копию строки с измененным значением символа в указанной позиции.

```
bool StringSetCharacter(
    string& string_var,           // строка
    int pos,                     // позиция
    ushort character            // символ
);
```

### Параметры

*string\_var*

[in][out] Страна.

*pos*

[in] Позиция символа в строке. Может быть от 0 до [StringLen\(text\)](#).

*character*

[in] Символьный код Unicode.

### Возвращаемое значение

В случае успешного выполнения возвращает true, иначе false. Для получения кода ошибки нужно вызвать функцию [GetLastError\(\)](#).

### Примечание

Если значение *pos* меньше [длины строки](#) и значение символьного кода = 0, то строка усекается (но [размер буфера](#), распределенного под строку остается неизменным). Длина строки становится равной значению *pos*.

Если значение параметра *pos* равняется значению длины строки, то указанный символ добавляется в конец строки, и таким образом длина строки увеличивается на единицу.

### Пример:

```
void OnStart()
{
    string str="0123456789";
    Print("before: str = ",str,",StringBufferLen(str) = ",
          StringBufferLen(str)," StringLen(str) = ",StringLen(str));
    //--- вставим нулевое значение посреди строки
    StringSetCharacter(str,6,0);
    Print(" after: str = ",str,",StringBufferLen(str) = ",
          StringBufferLen(str)," StringLen(str) = ",StringLen(str));
    //--- добавим символ в конец строки
    int size=StringLen(str);
    StringSetCharacter(str,size,'+');
    Print("addition: str = ",str,",StringBufferLen(str) = ",
          StringBufferLen(str)," StringLen(str) = ",StringLen(str));
}
```

/\* Результат

```
before: str = 0123456789 ,StringBufferLen(str) = 0   StringLen(str) = 10
after: str = 012345 ,StringBufferLen(str) = 16   StringLen(str) = 6
addition: str = 012345+ ,StringBufferLen(str) = 16   StringLen(str) = 7
*/
```

Смотри также

[StringBufferLen](#), [StringLen](#), [StringFill](#), [StringInit](#), [CharToString](#), [ShortToString](#), [CharArrayToString](#), [ShortArrayToString](#)

## StringSplit

Получает из указанной строки подстроки по заданному разделителю и возвращает количество полученных подстрок.

```
int StringSplit(
    const string string_value,           // строка для поиска подстрок
    const ushort separator,              // разделитель, по которому в строке будут искат...
    string & result[]                  // массив, переданный по ссылке, для получения р...
);
```

### Параметры

*string\_value*

[in] Стока, из которой необходимо получить подстроки. Сама строка при этом не изменяется.

*pos*

[in] Код символа разделителя. Для получения кода можно использовать функцию [StringGetCharacter\(\)](#).

*result[]*

[out] Массив строк, в который помещаются полученные подстроки.

### Возвращаемое значение

Количество полученных строк в массиве *result[]*. Если разделитель в переданной строке не найден, то в массив будет помещена только одна исходная строка.

Если строка *string\_value* пустая или NULL, то функция вернет ноль. В случае ошибки функция вернет -1. Для получения кода [ошибки](#) нужно вызвать функцию [GetLastError\(\)](#).

### Пример:

```
string to_split=_мама_мыла_раму_; // строка для разбивки на подстроки
string sep=_;
ushort u_sep; // код символа разделителя
string result[]; // массив для получения строк
//--- получим код разделителя
u_sep=StringGetCharacter(sep,0);
//--- разобьем строку на подстроки
int k=StringSplit(to_split,u_sep,result);
//--- выведем комментарий
PrintFormat("Получено строк: %d. Использован разделитель '%s' с кодом %d",k,sep,u_sep);
//--- теперь выведем все полученные строки
if(k>0)
{
    for(int i=0;i<k;i++)
    {
        PrintFormat("result[%d]=\"%s\"",i,result[i]);
    }
}
```

Смотри также

[StringReplace\(\)](#), [StringSubstr\(\)](#), [StringConcatenate\(\)](#)

## StringSubstr

Извлекает подстроку из текстовой строки, начинающейся с указанной позиции.

```
string StringSubstr(
    string string_value,           // строка
    int    start_pos,             // с какой позиции начать
    int    length=-1              // длина извлекаемой строки
);
```

### Параметры

*string\_value*

[in] Стока, из которой должна быть извлечена подстрока.

*start\_pos*

[in] Начальная позиция подстроки. Может быть от 0 до [StringLen\(text\)](#) -1.

*length=-1*

[in] Длина извлекаемой подстроки. Если значение параметра равно -1 либо параметр не задан, то будет извлекаться подстрока, начиная с указанной позиции и до конца строки.

### Возвращаемое значение

Копия извлеченной подстроки, если возможно, иначе возвращается пустая строка.

### Смотри также

[StringSplit](#), [StringFind](#), [StringGetCharacter](#)

## StringToLower

Преобразует все символы указанной строки в строчные (маленькие) по месту.

```
bool StringToLower(  
    string& string_var           // строка для обработки  
);
```

### Параметры

*string\_var*  
[in][out] Страна.

### Возвращаемое значение

В случае успешного выполнения возвращает true, иначе false. Для получения кода ошибки нужно вызвать функцию [GetLastError\(\)](#).

### Смотри также

[StringToUpper](#), [StringTrimLeft](#), [StringTrimRight](#)

## StringToUpper

Преобразует все символы указанной строки в прописные (большие) по месту.

```
bool StringToUpper(
    string& string_var           // строка для обработки
);
```

### Параметры

*string\_var*  
[in][out] Страна.

### Возвращаемое значение

В случае успешного выполнения возвращает true, иначе false. Для получения кода ошибки нужно вызвать функцию [GetLastError\(\)](#).

### Смотри также

[StringToLower](#), [StringTrimLeft](#), [StringTrimRight](#)

## StringTrimLeft

Удаляет символы перевода каретки, пробелы и символы табуляции с начала строки до первого значимого символа. Стока модифицируется по месту.

```
int StringTrimLeft(  
    string& string_var           // строка для обрезки  
);
```

### Параметры

*string\_var*  
[in][out] Стока, которая должна быть обрезана слева.

### Возвращаемое значение

Возвращает количество отрезанных символов.

### Смотри также

[StringTrimRight](#), [StringToLower](#), [StringToUpper](#)

## StringTrimRight

Удаляет символы перевода каретки, пробелы и символы табуляции от последнего значимого символа до конца строки. Стока модифицируется по месту.

```
int StringTrimRight(  
    string& string_var          // строка для обрезки  
);
```

### Параметры

*string\_var*  
[in][out] Стока, которая должна быть обрезана справа.

### Возвращаемое значение

Возвращает количество отрезанных символов.

### Смотри также

[StringTrimLeft](#), [StringToLower](#), [StringToUpper](#)

## 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).

Для организации счетчиков и таймеров высокого разрешения нужно использовать функцию `GetTickCount()`, которая выдает значения в миллисекундах.

Функция	Действие
<a href="#">TimeCurrent</a>	Возвращает последнее известное время сервера (время прихода последней котировки) в формате <code>datetime</code>
<a href="#">TimeTradeServer</a>	Возвращает расчетное текущее время торгового сервера
<a href="#">TimeLocal</a>	Возвращает локальное компьютерное время в формате <code>datetime</code>
<a href="#">TimeGMT</a>	Возвращает время GMT формате <code>datetime</code> с учетом перехода на зимнее или летнее время по локальному времени компьютера, на котором запущен клиентский терминал
<a href="#">TimeDaylightSavings</a>	Возвращает признак перехода на летнее /зимнее время
<a href="#">TimeGMTOffset</a>	Возвращает текущую разницу между временем GMT и локальным временем компьютера в секундах с учетом перехода на зимнее или летнее время
<a href="#">TimeToStruct</a>	Производит конвертацию из значения типа <code>datetime</code> в переменную типа структуры <code>MqlDateTime</code>
<a href="#">StructToTime</a>	Производит конвертацию из переменной типа структуры <code>MqlDateTime</code> в значение типа <code>datetime</code>

## TimeCurrent

Возвращает последнее известное время сервера, время прихода последней котировки по одному из выбранных в "Обзоре рынка" символов. В обработчике [OnTick\(\)](#) данная функция вернет время пришедшего обрабатываемого тика. В других случаях (например, вызов в [обработчиках](#) OnInit(), OnDeinit(), OnTimer() и так далее) это - [время прихода последней котировки](#) по любому символу, доступного в окне "Обзор рынка", то самое время, которое показано в заголовке этого окна. Значение времени формируется на торговом сервере и не зависит от настроек времени на компьютере пользователя. Существует 2 варианта функции.

### Вызов без параметров

```
datetime TimeCurrent();
```

### Вызов с параметром типа MqlDateTime

```
datetime TimeCurrent(
    MqlDateTime& dt_struct           // переменная типа структуры
);
```

### Параметры

*dt\_struct*  
 [out] Переменная типа структуры [MqlDateTime](#).

### Возвращаемое значение

Значение типа [datetime](#)

### Примечание

Если в качестве параметра была передана переменная типа структуры MqlDateTime, то она заполняется соответствующим образом.

Для организации счетчиков и таймеров высокого разрешения нужно использовать функцию [GetTickCount\(\)](#), которая выдает значения в миллисекундах.

При работе в тестере стратегий время последней котировки TimeCurrent() моделируется в соответствии с историческими данными.

## TimeTradeServer

Возвращает расчетное текущее время торгового сервера. В отличие от функции [TimeCurrent\(\)](#), расчет значения времени производится в клиентском терминале и зависит от настроек времени на компьютере пользователя. Существует 2 варианта функции.

### Вызов без параметров

```
datetime TimeTradeServer();
```

### Вызов с параметром типа MqlDateTime

```
datetime TimeTradeServer(  
    MqlDateTime& dt_struct // переменная типа структуры  
);
```

### Параметры

*dt\_struct*  
[out] Переменная типа структуры [MqlDateTime](#).

### Возвращаемое значение

Значение типа [datetime](#)

### Примечание

Если в качестве параметра была передана переменная типа структуры [MqlDateTime](#), то она заполняется соответствующим образом.

Для организации счетчиков и таймеров высокого разрешения нужно использовать функцию [GetTickCount\(\)](#), которая выдает значения в миллисекундах.

При работе в тестере стратегий время `TimeTradeServer()` моделируется в соответствии с историческими данными и всегда равно [TimeCurrent\(\)](#).

## TimeLocal

Возвращает локальное время компьютера, на котором запущен клиентский терминал. Существует 2 варианта функции.

### Вызов без параметров

```
datetime TimeLocal();
```

### Вызов с параметром типа MqlDateTime

```
datetime TimeLocal(  
    MqlDateTime& dt_struct // переменная типа структуры  
) ;
```

### Параметры

*dt\_struct*  
[out] Переменная типа структуры [MqlDateTime](#).

### Возвращаемое значение

Значение типа [datetime](#)

### Примечание

Если в качестве параметра была передана переменная типа структуры [MqlDateTime](#), то она заполняется соответствующим образом.

Для организации счетчиков и таймеров высокого разрешения нужно использовать функцию [GetTickCount\(\)](#), которая выдает значения в миллисекундах.

При работе в тестере стратегий локальное время [TimeLocal\(\)](#) всегда равно моделируемому серверному времени [TimeCurrent\(\)](#).

## TimeGMT

Возвращает время GMT, которое вычисляется с учётом перехода на зимнее или летнее время по локальному времени компьютера, на котором запущен клиентский терминал. Существует 2 варианта функции.

### Вызов без параметров

```
datetime TimeGMT();
```

### Вызов с параметром типа MqlDateTime

```
datetime TimeGMT(  
    MqlDateTime& dt_struct // переменная типа структуры  
);
```

### Параметры

*dt\_struct*  
[out] Переменная типа структуры [MqlDateTime](#).

### Возвращаемое значение

Значение типа [datetime](#)

### Примечание

Если в качестве параметра была передана переменная типа структуры [MqlDateTime](#), то она заполняется соответствующим образом.

Для организации счетчиков и таймеров высокого разрешения нужно использовать функцию [GetTickCount\(\)](#), которая выдает значения в миллисекундах.

При работе в тестере стратегий время TimeGMT() всегда равно моделируемому серверному времени [TimeTradeServer\(\)](#).

## TimeDaylightSavings

Возвращает поправку на летнее время в секундах, если был произведен переход на летнее время.  
Зависит от настроек времени на компьютере пользователя.

```
int TimeDaylightSavings();
```

### Возвращаемое значение

Если был произведен переход на зимнее (стандартное) время, то возвращается 0.

## TimeGMTOffset

Возвращает текущую разницу между временем GMT и локальным временем компьютера в секундах с учетом перехода на зимнее или летнее время. Зависит от настроек времени на компьютере пользователя.

```
int TimeGMTOffset();
```

### Возвращаемое значение

Значение типа int, представляющее текущую разницу между [временем GMT](#) и локальным временем компьютера [TimeLocal\(\)](#) и в секундах.

```
TimeGMTOffset() = TimeGMT() - TimeLocal()
```

## TimeToStruct

Производит конвертацию из значения типа datetime (количество секунд с 01.01.1970) в переменную типа структуры [MqlDateTime](#).

```
bool TimeToStruct(
    datetime      dt,           // дата и время
    MqlDateTime& dt_struct     // структура для принятия значений
);
```

### Параметры

*dt*

[in] Значение даты для конвертации.

*dt\_struct*

[out] Переменная структуры типа [MqlDateTime](#).

### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

## StructToTime

Производит конвертацию из переменной типа структуры [MqlDateTime](#) в значение типа [datetime](#) и возвращает полученное значение.

```
datetime StructToTime(  
    MqlDateTime& dt_struct // структура даты и времени  
) ;
```

### Параметры

*dt\_struct*  
[in] Переменная структуры типа [MqlDateTime](#).

### Возвращаемое значение

Значение типа [datetime](#), содержащее количество секунд с 01.01.1970.

## Информация о счете

Функции, возвращающие параметры текущего счета.

Функция	Действие
<a href="#"><u>AccountInfoDouble</u></a>	Возвращает значение типа double соответствующего свойства счета
<a href="#"><u>AccountInfoInteger</u></a>	Возвращает значение целочисленного типа (bool,int или long) соответствующего свойства счета
<a href="#"><u>AccountInfoString</u></a>	Возвращает значение типа string соответствующего свойства счета

## AccountInfoDouble

Возвращает значение соответствующего свойства счета.

```
double AccountInfoDouble(
    ENUM_ACCOUNT_INFO_DOUBLE property_id           // идентификатор свойства
);
```

### Параметры

*property\_id*  
 [in] Идентификатор свойства. Значение может быть одним из значений [ENUM\\_ACCOUNT\\_INFO\\_DOUBLE](#).

### Возвращаемое значение

Значение типа [double](#).

### Пример:

```
void OnStart()
{
//--- выведем всю информацию, доступную из функции AccountInfoDouble()
printf("ACCOUNT_BALANCE = %G", AccountInfoDouble(ACCOUNT_BALANCE));
printf("ACCOUNT_CREDIT = %G", AccountInfoDouble(ACCOUNT_CREDIT));
printf("ACCOUNT_PROFIT = %G", AccountInfoDouble(ACCOUNT_PROFIT));
printf("ACCOUNT_EQUITY = %G", AccountInfoDouble(ACCOUNT_EQUITY));
printf("ACCOUNT_MARGIN = %G", AccountInfoDouble(ACCOUNT_MARGIN));
printf("ACCOUNT_MARGIN_FREE = %G", AccountInfoDouble(ACCOUNT_MARGIN_FREE));
printf("ACCOUNT_MARGIN_LEVEL = %G", AccountInfoDouble(ACCOUNT_MARGIN_LEVEL));
printf("ACCOUNT_MARGIN_SO_CALL = %G", AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL));
printf("ACCOUNT_MARGIN_SO_SO = %G", AccountInfoDouble(ACCOUNT_MARGIN_SO_SO));
}
```

### Смотри также

[SymbolInfoDouble](#), [SymbolInfoString](#), [SymbolInfoInteger](#), [PrintFormat](#)

## AccountInfoInteger

Возвращает значение соответствующего свойства счета.

```
long AccountInfoInteger(
    ENUM_ACCOUNT_INFO_INTEGER property_id           // идентификатор свойства
);
```

### Параметры

*property\_id*  
 [in] Идентификатор свойства. Значение может быть одним из значений [ENUM\\_ACCOUNT\\_INFO\\_INTEGER](#).

### Возвращаемое значение

Значение типа [long](#).

### Примечание

Свойство должно быть одного из типов [bool](#), [int](#) или [long](#).

### Пример:

```
void OnStart()
{
//--- выведем всю информацию, доступную из функции AccountInfoInteger()
printf("ACCOUNT_LOGIN = %d",AccountInfoInteger(ACCOUNT_LOGIN));
printf("ACCOUNT_LEVERAGE = %d",AccountInfoInteger(ACCOUNT_LEVERAGE));
bool thisAccountTradeAllowed=AccountInfoInteger(ACCOUNT_TRADE_ALLOWED);
bool EATradeAllowed=AccountInfoInteger(ACCOUNT_TRADE_EXPERT);
ENUM_ACCOUNT_TRADE_MODE tradeMode=(ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(ACCOUNT_TRADE_MODE);
ENUM_ACCOUNT_STOPOUT_MODE stopOutMode=(ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInteger(ACCOUNT_STOPOUT_MODE);

//--- сообщим о возможности совершения торговых операций
if(thisAccountTradeAllowed)
    Print("Торговля для данного счета разрешена");
else
    Print("Торговля для данного счета запрещена!");

//--- выясним - можно ли торговать на данном счету экспертами
if(EATradeAllowed)
    Print("Торговля советниками для данного счета разрешена");
else
    Print("Торговля советниками для данного счета запрещена");

//--- выясним тип счета
switch(tradeMode)
{
    case(ACCOUNT_TRADE_MODE_DEMO):
        Print("Это демо счет");
        break;
```

```
case(ACCOUNT_TRADE_MODE_CONTEST) :
    Print("Это конкурсный счет");
    break;
default:Print("Это реальный счет!");
}

//--- выясним режим задания уровня StopOut
switch(stopOutMode)
{
    case(ACCOUNT_STOPOUT_MODE_PERCENT):
        Print("Уровень StopOut задается в процентах");
        break;
    default:Print("Уровень StopOut задается в денежном выражении");
}
}
```

Смотри также

[Информация о счете](#)

## AccountInfoString

Возвращает значение соответствующего свойства счета.

```
string AccountInfoString(
    ENUM_ACCOUNT_INFO_STRING property_id           // идентификатор свойства
);
```

### Параметры

*property\_id*  
 [in] Идентификатор свойства. Значение может быть одним из значений [ENUM\\_ACCOUNT\\_INFO\\_STRING](#).

### Возвращаемое значение

Значение типа [string](#).

### Пример:

```
void OnStart()
{
//--- выведем всю информацию, доступную из функции AccountInfoString()
Print("Имя брокера = ",AccountInfoString(ACCOUNT_COMPANY));
Print("Валюта депозита = ",AccountInfoString(ACCOUNT_CURRENCY));
Print("Имя клиента = ",AccountInfoString(ACCOUNT_NAME));
Print("Название торгового сервера = ",AccountInfoString(ACCOUNT_SERVER));
}
```

### Смотри также

[Информация о счете](#)

## Проверка состояния

Функции, возвращающие параметры текущего состояния клиентского терминала

Функция	Действие
<a href="#">GetLastError</a>	Возвращает значение последней ошибки
<a href="#">IsStopped</a>	Возвращает true, если поступила команда завершить выполнение mql5-программы
<a href="#">UninitializeReason</a>	Возвращает код причины deinициализации
<a href="#">TerminalInfoInteger</a>	Возвращает значение целого типа соответствующего свойства окружения mql5-программы
<a href="#">TerminalInfoDouble</a>	Возвращает значение типа double соответствующего свойства окружения mql5-программы
<a href="#">TerminalInfoString</a>	Возвращает значение типа string соответствующего свойства окружения mql5-программы
<a href="#">MQLInfoInteger</a>	Возвращает значение целого типа соответствующего свойства запущенной mql5-программы
<a href="#">MQLInfoString</a>	Возвращает значение типа string соответствующего свойства запущенной mql5-программы
<a href="#">Symbol</a>	Возвращает имя символа текущего графика.
<a href="#">Period</a>	Возвращает значение таймфрейма текущего графика
<a href="#">Digits</a>	Возвращает количество десятичных знаков после запятой, определяющее точность измерения цены символа текущего графика
<a href="#">Point</a>	Возвращает размер пункта текущего инструмента в валюте котировки.

## GetLastError

Возвращает содержимое системной переменной `_LastError`.

```
int GetLastError();
```

### Возвращаемое значение

Возвращает значение последней [ошибки](#), произошедшей во время исполнения MQL5-программы.

### Примечание

После вызова функции содержимое переменной `_LastError` не обнуляется. Чтобы обнулить эту переменную, необходимо вызвать функцию [ResetLastError\(\)](#)

### Смотри также

[Коды возврата торгового сервера](#)

## IsStopped

Проверяет принудительное завершение работы mq5-программы.

```
bool IsStopped();
```

### Возвращаемое значение

Возвращает true, если в системной переменной [\\_StopFlag](#) содержится значение, отличное от 0. Ненулевое значение записывается в переменную [\\_StopFlag](#), если поступила команда завершить выполнение mq5-программы. В этом случае необходимо как можно быстрее завершить работу программы, в противном случае программа будет завершена принудительно извне через 3 секунды.

## UninitializeReason

Возвращает код [причины deinициализации](#).

```
int UninitializeReason();
```

### Возвращаемое значение

Возвращает значение переменной [\\_UninitReason](#), которое формируется перед вызовом функции [OnDeinit\(\)](#). Значение зависит от причины, приведшей к deinициализации.

## TerminalInfoInteger

Возвращает значение соответствующего свойства окружения MQL5-программы.

```
int TerminalInfoInteger(  
    int property_id // идентификатор свойства  
) ;
```

### Параметры

*property\_id*  
[in] Идентификатор свойства. Может быть одним из значений перечисления [ENUM\\_TERMINAL\\_INFO\\_INTEGER](#).

### Возвращаемое значение

Значение типа int.

## TerminalInfoDouble

Возвращает значение соответствующего свойства окружения mq5-программы.

```
double TerminalInfoDouble(
    int property_id          // идентификатор свойства
);
```

### Параметры

*property\_id*  
[in] Идентификатор свойства. Может быть одним из значений перечисления [ENUM\\_TERMINAL\\_INFO\\_DOUBLE](#).

### Возвращаемое значение

Значение типа double.

## TerminalInfoString

Функция возвращает значение соответствующего свойства окружения MQL5-программы. Свойство должно быть типа string

```
string TerminalInfoString(  
    int property_id // идентификатор свойства  
);
```

### Параметры

*property\_id*  
[in] Идентификатор свойства. Может быть одним из значений перечисления [ENUM\\_TERMINAL\\_INFO\\_STRING](#).

### Возвращаемое значение

Значение типа string.

## MQLInfoInteger

Возвращает значение соответствующего свойства запущенной MQL5-программы.

```
int MQLInfoInteger(
    int property_id          // идентификатор свойства
);
```

### Параметры

*property\_id*  
[in] Идентификатор свойства. Может быть одним из значений перечисления [ENUM\\_MQL\\_INFO\\_INTEGER](#).

### Возвращаемое значение

Значение типа int.

## MQLInfoString

Возвращает значение соответствующего свойства запущенной MQL5-программы.

```
string MQLInfoString(  
    int property_id // идентификатор свойства  
) ;
```

### Параметры

*property\_id*  
[in] Идентификатор свойства. Может быть одним из значений перечисления [ENUM\\_MQL\\_INFO\\_STRING](#).

### Возвращаемое значение

Значение типа string.

## Symbol

Возвращает имя символа текущего графика.

```
string Symbol();
```

### Возвращаемое значение

Значение системной переменной [Symbol](#), в которой хранится имя символа текущего графика.

## Period

Возвращает значение таймфрейма текущего графика.

```
ENUM_TIMEFRAMES Period();
```

### Возвращаемое значение

Содержимое переменной [Period](#), в которой хранится значение таймфрейма текущего графика.  
Значение может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#).

### Смотри также

[PeriodSeconds](#), [Периоды графиков](#), [Дата и время](#), , [Видимость объектов](#)

## Digits

Возвращает количество десятичных знаков после запятой, определяющее точность измерения цены символа текущего графика.

```
int Digits();
```

### Возвращаемое значение

Значение переменной [Digits](#), в которой хранится количество десятичных знаков после запятой, определяющее точность измерения цены символа текущего графика.

## Point

Возвращает размер пункта текущего инструмента в валюте котировки.

```
double Point();
```

### Возвращаемое значение

Значение переменной [Point](#), в которой хранится размер пункта текущего инструмента в валюте котировки.

## Обработка событий

В языке MQL5 предусмотрена обработка некоторых [предопределенных событий](#). Функции для обработки этих событий должны быть определены в программе MQL5: имя функции, тип возвращаемого значения, состав параметров (если они есть) и их типы должны строго соответствовать описанию функции-обработчика события.

Именно по типу возвращаемого значения и по типам параметров обработчик событий клиентского терминала идентифицирует функции, обрабатывающие то или иное событие. Если у соответствующей функции указаны иные, не соответствующие нижеследующим описаниям, параметры или указан иной тип возвращаемого значения, то такая функция не будет использоваться для обработки события.

Функция	Описание
<a href="#">OnStart</a>	Вызывается в скрипте при наступлении события <a href="#">Start</a> для выполнения действий, заложенных в скрипт
<a href="#">OnInit</a>	Вызывается в индикаторах и экспертах при наступлении события <a href="#">Init</a> для инициализации запущенной MQL5-программы
<a href="#">OnDeinit</a>	Вызывается в индикаторах и экспертах при наступлении события <a href="#">Deinit</a> для деинициализации запущенной MQL5-программы
<a href="#">OnTick</a>	Вызывается в экспертах при наступлении события <a href="#">NewTick</a> для обработки новой котировки
<a href="#">OnCalculate</a>	Вызывается в индикаторах при наступлении события <a href="#">Calculate</a> для обработки изменения ценовых данных
<a href="#">OnTimer</a>	Вызывается в индикаторах и экспертах при наступлении периодического события <a href="#">Timer</a> , которое с заданным интервалом времени генерируется терминалом
<a href="#">OnTrade</a>	Вызывается в экспертах при наступлении события <a href="#">Trade</a> , которое генерируется при завершении торговой операции на торговом сервере
<a href="#">OnTradeTransaction</a>	Вызывается в экспертах при наступлении события <a href="#">TradeTransaction</a> для обработки результатов выполнения торгового запроса
<a href="#">OnBookEvent</a>	Вызывается в экспертах при наступлении события <a href="#">BookEvent</a> для обработки изменений в стакане заявок

<a href="#">OnChartEvent</a>	Вызывается в индикаторах и экспертах при наступлении события <a href="#">ChartEvent</a> для обработки изменений графика, вызванных действиями пользователя или работой MQL5-программ
<a href="#">OnTester</a>	Вызывается в экспертах при наступлении события <a href="#">Tester</a> для выполнения необходимых действий по окончании тестирования
<a href="#">OnTesterInit</a>	Вызывается в экспертах при наступлении события <a href="#">TesterInit</a> для выполнения необходимых действий перед началом оптимизации
<a href="#">OnTesterDeinit</a>	Вызывается в экспертах при наступлении события <a href="#">TesterDeinit</a> для выполнения необходимых действий по окончании оптимизации эксперта
<a href="#">OnTesterPass</a>	Вызывается в экспертах при наступлении события <a href="#">TesterPass</a> для обработки поступления нового фрейма данных во время оптимизации эксперта

Клиентский терминал отсылает возникающие события в соответствующие открытые графики. Также события могут генерироваться графиками ([события графика](#)) либо mq5-программами ([пользовательские события](#)). Генерацию событий создания и удаления графических объектов на графике можно включать и отключать заданием свойств графика [CHART\\_EVENT\\_OBJECT\\_CREATE](#) и [CHART\\_EVENT\\_OBJECT\\_DELETE](#). Каждая mq5-программа и каждый график имеют свою собственную очередь событий, куда складываются все вновь поступающие события.

Программа получает события только от графика, на котором она запущена. Все события обрабатываются одно за другим в порядке поступления. Если в очереди уже есть событие [NewTick](#) либо это событие находится в состоянии обработки, то новое событие NewTick в очередь mq5-программы не ставится. Аналогично, если в очереди mq5-программы уже находится событие [ChartEvent](#) или такое событие обрабатывается, то новое событие такого типа не ставится в очередь. Обработка событий таймера производится по такой же схеме - если в очереди находится или уже обрабатывается событие [Timer](#), то новое событие таймера не ставится в очередь.

Очереди событий имеют ограниченный, но достаточный размер, поэтому переполнение очереди для корректно написанной программы маловероятно. При переполнении очереди новые события отбрасываются без постановки в очередь.

Крайне не рекомендуется использовать бесконечные циклы для обработки событий. Исключением из этого правила могут быть только скрипты, которые обрабатывают одно единственное событие [Start](#).

[Библиотеки](#) не обрабатывают никаких событий.

## OnStart

Вызывается в скриптах и сервисах при наступлении события [Start](#). Функция предназначена для однократного выполнения действий, заложенных в программу. Существуют два варианта функции.

### Версия с возвратом результата

```
int OnStart(void);
```

### Возвращаемое значение

Значение типа [int](#), которое выводится в закладку "Журнал".

После завершения скрипта в журнале терминала будет создана запись вида "script имя\_скрипта removed (result code N)", где N и есть то значение, которое вернула функция OnStart().

После завершения сервиса в журнале терминала будет создана запись вида "service имя\_сервиса stopped (result code N)", где N и есть то значение, которое вернула функция OnStart().

Приоритетным является использование вызова OnStart() с возвратом результата выполнения, так как этот способ позволяет не только выполнить скрипт или сервис, но и вернуть код ошибки или другую полезную информацию для анализа результата выполнения работы программы.

**Версия без возврата результата** оставлена только для совместимости со старыми кодами. Не рекомендуется к использованию

```
void OnStart(void);
```

### Примечание

OnStart() является единственной функцией для обработки событий в скриптах и сервисах, другие события в эти программы не посыпаются. В свою очередь событие [Start](#) не посыпается экспертам и пользовательским индикаторам.

### Пример скрипта:

```
//--- макросы для работы с цветом
#define XRGB(r,g,b)      (0xFF000000 | (uchar(r)<<16) | (uchar(g)<<8) | uchar(b))
#define GETRGB(clr)       ((clr)&0xFFFFFFFF)
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- установим цвет падающей свечи
Comment("Установим цвет падающей свечи");
ChartSetInteger(0,CHART_COLOR_CANDLE_BEAR,GetRandomColor());
ChartRedraw(); // немедленно обновим график без ожидания нового тика
Sleep(1000);   // сделаем паузу в 1 секунду, чтобы можно было увидеть изменения
//--- установим цвет растущей свечи
Comment("Установим цвет растущей свечи");
ChartSetInteger(0,CHART_COLOR_CANDLE_BULL,GetRandomColor());
ChartRedraw();
```

```

Sleep(1000);

//--- установим цвет фона
Comment("Установим цвет фона");
ChartSetInteger(0,CHART_COLOR_BACKGROUND,GetRandomColor());
ChartRedraw();
Sleep(1000);

//--- установим цвет линии Ask
Comment("Установим цвет линии Ask");
ChartSetInteger(0,CHART_COLOR_ASK,GetRandomColor());
ChartRedraw();
Sleep(1000);

//--- установим цвет линии Bid
Comment("Установим цвет линии Bid");
ChartSetInteger(0,CHART_COLOR_BID,GetRandomColor());
ChartRedraw();
Sleep(1000);

//--- установим цвет падающего бара и окантовки падающей свечи
Comment("Установим цвет падающего бара и окантовки падающей свечи");
ChartSetInteger(0,CHART_COLOR_CHART_DOWN,GetRandomColor());
ChartRedraw();
Sleep(1000);

//--- установим цвет линии графика и свечей типа "Доджи"
Comment("Установим цвет линии графика и свечей типа Доджи");
ChartSetInteger(0,CHART_COLOR_CHART_LINE,GetRandomColor());
ChartRedraw();
Sleep(1000);

//--- установим цвет растущего бара и окантовки растущей свечи
Comment("Установим цвет растущего бара и окантовки растущей свечи");
ChartSetInteger(0,CHART_COLOR_CHART_UP,GetRandomColor());
ChartRedraw();
Sleep(1000);

//--- установим цвет осей, шкалы и строки OHLC
Comment("Установим цвет осей, шкалы и строки OHLC");
ChartSetInteger(0,CHART_COLOR_FOREGROUND,GetRandomColor());
ChartRedraw();
Sleep(1000);

//--- установим цвет сетки
Comment("Установим цвет сетки");
ChartSetInteger(0,CHART_COLOR_GRID,GetRandomColor());
ChartRedraw();
Sleep(1000);

//--- установим цвет цены Last
Comment("Установим цвет цены Last");
ChartSetInteger(0,CHART_COLOR_LAST,GetRandomColor());
ChartRedraw();
Sleep(1000);

//--- установим цвет уровней ордеров Stop Loss и Take Profit
Comment("Установим цвет уровней ордеров Stop Loss и Take Profit");
ChartSetInteger(0,CHART_COLOR_STOP_LEVEL,GetRandomColor());

```

```
ChartRedraw();
Sleep(1000);
//--- установим цвет объемов и уровней открытия позиций
Comment("Установим цвет объемов и уровней открытия позиций");
ChartSetInteger(0,CHART_COLOR_VOLUME,GetRandomColor());
ChartRedraw();
}

//+-----+
//| Возвращает цвет, сгенерированный случайным образом |+
//+-----+
color GetRandomColor()
{
    color clr=(color)GETRGB(XRGB(rand()%255,rand()%255,rand()%255));
    return clr;
}
```

#### Смотри также

[Функции обработки событий](#), [Выполнение программ](#), [События клиентского терминала](#)

## OnInit

Вызывается в индикаторах и экспертах при наступлении события [Init](#). Функция предназначена для инициализации запущенной MQL5-программы. Существуют два варианта функции.

### Версия с возвратом результата

```
int OnInit(void);
```

### Возвращаемое значение

Значение типа [int](#), ноль означает успешную инициализацию.

Приоритетным является использование вызова `OnInit()` с возвратом результата выполнения, так как этот способ позволяет не только выполнить инициализацию программы, но и вернуть код ошибки в случае досрочного прекращения программы.

**Версия без возврата результата** оставлена только для совместимости со старыми кодами. Не рекомендуется к использованию

```
void OnInit(void);
```

### Примечание

Событие `Init` генерируется сразу после загрузки эксперта или индикатора, для скриптов это событие не генерируется. Функция `OnInit()` используется для инициализации программы MQL5. Если `OnInit()` имеет возвращаемое значение типа [int](#), то ненулевой код возврата означает неудачную инициализацию и генерирует событие [Deinit](#) с кодом причины деинициализации [REASON\\_INITFAILED](#).

Функция `OnInit()` типа [void](#) всегда означает удачную инициализацию и не рекомендуется к использованию.

При [оптимизации входных параметров](#) эксперта рекомендуется в качестве кода возврата использовать значения из перечисления [ENUM\\_INIT\\_RETCODE](#). Эти значения предназначены для организации управления ходом оптимизации, в том числе для выбора наиболее подходящих [агентов тестирования](#). Прямо при инициализации эксперта еще до запуска самого тестирования можно запросить информацию о конфигурации и ресурсах агента (количество ядер, объем свободной памяти и т.д.) с помощью функции [TerminalInfoInteger\(\)](#). На основе полученной информации можно либо разрешить использовать данный агент тестирования, либо отказаться от него при оптимизации данного эксперта.

Идентификатор	Описание
INIT_SUCCEEDED	Инициализация прошла успешно, тестирование эксперта можно продолжать. Этот код означает то же самое, что и нулевое значение - инициализация эксперта в тестере прошла успешно.
INIT_FAILED	Неудачная инициализация, тестирование нет смысла продолжать из-за неустранимых ошибок. Например, не удалось создать

	<p>индикатор, необходимый для работы эксперта.</p> <p>Возврат этого значения означает то же самое, что и возврат значения, отличного от нуля, - инициализация эксперта в тестере прошла неудачно.</p>
INIT_PARAMETERS_INCORRECT	<p>Предназначен для обозначения программистом некорректного набора входных параметров, в общей таблице оптимизации строка результата с таким кодом возврата будет подсвечена красным цветом.</p> <p>Тестирование для данного набора параметров эксперта не будет выполняться, агент свободен для получения нового задания.</p> <p>При получении этого значения тестер стратегий гарантированно не будет передавать данное задание другим агентам для повторного выполнения.</p>
INIT_AGENT_NOT_SUITABLE	<p>Ошибка в работе программы при инициализации не возникло, но по каким-то причинам данный агент не подходит для проведения тестирования. Например, недостаточно оперативной памяти, нет <a href="#">поддержки OpenCL</a> и так далее.</p> <p>После возврата этого кода агент больше не будет получать заданий до самого конца <a href="#">данной оптимизации</a>.</p>

Использование [OnInit\(\)](#) с возвратом INIT\_FAILED/INIT\_PARAMETERS\_INCORRECT в тестере имеет ряд особенностей, которые необходимо учитывать при оптимизации советников:

- набор параметров, для которого OnInit() вернула INIT\_PARAMETERS\_INCORRECT считается непригодным для тестирования и не будет использован для получения следующей популяции при [генетической оптимизации](#). Если таких "забракованных" наборов параметров будет слишком много, то это может привести к некорректным результатам поиска оптимальных параметров советника. Алгоритм поиска предполагает, что функция [критерия оптимизации](#) является гладкой и не имеет разрывов на всём множестве входных параметров.
- если OnInit() вернула INIT\_FAILED, то это означает, что тестирование нельзя начинать и советник выгружается из памяти агента. Для выполнения следующего прохода на новом наборе параметров будет произведена повторная загрузка советника. Это потребует намного больше времени для запуска тестирования следующего прохода оптимизации, чем в случае вызова TesterStop().

### Пример функции OnInit() для эксперта

```
//--- input parameters
 ma_period=20; // период скользящей средней

//--- хендл индикатора, который используется в советнике
```

```

int indicator_handle;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- проверим на корректность значение ma_period
if(ma_period<=0)
{
    PrintFormat("Недопустимое значение входного параметра ma_period: %d",ma_period);
    return (INIT_PARAMETERS_INCORRECT);
}
//--- при оптимизации
if(MQLInfoInteger(MQL_OPTIMIZATION))
{
//--- проверим объем доступной оперативной памяти для агента
int available_memory_mb=TerminalInfoInteger(TERMINAL_MEMORY_TOTAL);
if(available_memory_mb<2000)
{
    PrintFormat("Недостаточный объем памяти для агента тестирования: %d MB",
               available_memory_mb);
    return (INIT_AGENT_NOT_SUITABLE);
}
}
//--- проверим наличие индикатора
indicator_handle=iCustom(_Symbol,_Period,"My_Indicator",ma_period);
if(indicator_handle==INVALID_HANDLE)
{
    PrintFormat("Не удалось создать хендл индикатора My_Indicator. Код ошибки %d",
               GetLastError());
    return (INIT_FAILED);
}
//--- инициализация эксперта прошла удачно
return(INIT_SUCCEEDED);
}

```

#### Смотри также

[OnDeinit](#), [TesterHideIndicators](#), [Функции обработки событий](#), [Выполнение программ](#), [События клиентского терминала](#), [Инициализация переменных](#), [Создание и уничтожение объектов](#)

## OnDeinit

Вызывается в индикаторах и экспертах при наступлении события [Deinit](#). Функция предназначена для деинициализации запущенной MQL5-программы.

```
void OnDeinit(
    const int reason           // код причины деинициализации
);
```

### Параметры

*reason*  
 [in] Код причины деинициализации.

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

Событие Deinit генерируется для экспертов и индикаторов в следующих случаях:

- перед переинициализацией в связи со сменой символа или периода графика, к которому прикреплена mql5-программа;
- перед переинициализацией в связи со сменой [входных параметров](#);
- перед выгрузкой mql5-программы.

Параметр *reason* может принимать следующие значения:

Константа	Значение	Описание
REASON_PROGRAM	0	Эксперт прекратил свою работу, вызвав функцию <a href="#">ExpertRemove()</a>
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 либо произошло переподключение к торговому серверу вследствие изменения настроек счета

REASON_TEMPLATE	7	Применен другой шаблон графика
REASON_INITFAILED	8	Обработчик <a href="#">OnInit()</a> вернул ненулевое значение
REASON_CLOSE	9	Терминал был закрыт

Коды причины деинициализации [эксперта](#) можно также получить функцией [UninitializeReason\(\)](#) или из предопределенной переменной [\\_UninitReason](#).

### Пример функций OnInit() и OnDeinit() для эксперта

```

input int fake_parameter=3;           // бесполезный параметр
//+-----+
//| Expert initialization function   |
//+-----+
int OnInit()
{
    //--- Получим номер билда, в котором скомпилирована программа
    Print(__FUNCTION__," Build #",__MQLBUILD__);
    //--- Код причины перезагрузки можно получать и в OnInit()
    Print(__FUNCTION__," При перезагрузке эксперта можно получить код причины деинициа
    //--- Первый способ получить код причины деинициализации
    Print(__FUNCTION__," _UninitReason = ",getUninitReasonText(_UninitReason));
    //--- Второй способ получить код причины деинициализации
    Print(__FUNCTION__," UninitializeReason() = ",getUninitReasonText(UninitializeReason));
    //---

    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
    //--- Первый способ получить код причины деинициализации
    Print(__FUNCTION__," Код причины деинициализации = ",reason);
    //--- Второй способ получить код причины деинициализации
    Print(__FUNCTION__," _UninitReason = ",getUninitReasonText(_UninitReason));
    //--- Третий способ получить код причины деинициализации
    Print(__FUNCTION__," UninitializeReason() = ",getUninitReasonText(UninitializeReason));
}

//+-----+
//| Возвращает текстовое описание причины деинициализации |
//+-----+
string getUninitReasonText(int reasonCode)
{
    string text="";
    //---
    switch(reasonCode)

```

```
{  
    case REASON_ACCOUNT:  
        text="Account was changed";break;  
    case REASON_CHARTCHANGE:  
        text="Symbol or timeframe was changed";break;  
    case REASON_CHARTCLOSE:  
        text="Chart was closed";break;  
    case REASON_PARAMETERS:  
        text="Input-parameter was changed";break;  
    case REASON_RECOMPILE:  
        text="Program "+__FILE__+" was recompiled";break;  
    case REASON_REMOVE:  
        text="Program "+__FILE__+" was removed from chart";break;  
    case REASON_TEMPLATE:  
        text="New template was applied to chart";break;  
    default:text="Another reason";  
}  
//---  
return text;  
}
```

#### Смотри также

[OnInit](#), [Функции обработки событий](#), [Выполнение программ](#), [События клиентского терминала](#), [Причины deinициализации](#), [Область видимости и время жизни переменных](#), [Создание и уничтожение объектов](#)

## OnTick

Вызывается в экспертах при наступлении события [NewTick](#) для обработки новой котировки.

```
void OnTick(void);
```

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

Событие [NewTick](#) генерируется только для экспертов при поступлении нового тика по символу, к графику которого прикреплен эксперт. Функцию `OnTick()` бесполезно определять в пользовательском индикаторе или скрипте, поскольку событие [NewTick](#) для них не генерируется.

Событие [Tick](#) генерируется только для экспертов, но это не означает, что эксперты обязаны иметь функцию `OnTick()`, так как для экспертов генерируются не только события [NewTick](#), но и события [Timer](#), [BookEvent](#) и [ChartEvent](#).

Все события обрабатываются одно за другим в порядке поступления. Если в очереди уже есть событие [NewTick](#) либо это событие находится в состоянии обработки, то новое событие [NewTick](#) в очередь mql5-программы не ставится.

Событие [NewTick](#) генерируется независимо от того, запрещена или разрешена автоматическая торговля (кнопка "Авто-торговля"). Запрет автоматической торговли означает только запрет на отправку торговых запросов из эксперта, работа эксперта при этом не прекращается.

Запрет автоматической торговли путем нажатия кнопки "Авто-торговля" не прерывает текущее выполнение функции `OnTick()`.

### Пример эксперта, в котором вся торговая логика помещена в функцию `OnTick()`

```
//+-----+
//|                                             TradeByATR.mq5 |
//|                                         Copyright 2018, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Пример советника, торгующего в направлении \"взрывной\" свечи"
#property description "\"Взрывная\" свеча имеет тело размером более k*ATR"
#property description "Параметр \"revers\" переворачивает направление сигнала"

input double lots=0.1;           // объем в лотах
input double kATR=3;             // длина сигнальной свечи в ATR
input int    ATRperiod=20;        // период индикатора ATR
input int    holdbars=8;          // сколько баров удерживаем позицию
input int    slippage=10;          // допустимое проскальзывание
input bool   revers=false;       // переворачиваем сигнал?
input ulong  EXPERT_MAGIC=0;     // MagicNumber эксперта
```

```

//--- для хранения хендла индикатора ATR
int atr_handle;
//--- здесь будем хранить последние значения ATR и тела свечи
double last_atr,last_body;
datetime lastbar_timeopen;
double trade_lot;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//--- инициализируем глобальные переменные
last_atr=0;
last_body=0;
//--- установим правильный объем
double min_lot=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
trade_lot=lots>min_lot? lots:min_lot;
//--- создадим хендл индикатора ATR
atr_handle=iATR(_Symbol,_Period,ATRperiod);
if(atr_handle==INVALID_HANDLE)
{
PrintFormat("%s: не удалось создать iATR, код ошибки %d",__FUNCTION__,GetLastError());
return(INIT_FAILED);
}
//--- успешная инициализация эксперта
return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- сообщим код завершения работы эксперта
Print(__FILE__,": Код причины деинициализации = ",reason);
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{
//--- торговый сигнал
static int signal=0; // +1 означает сигнал на покупку, -1 означает сигнал на продажу
//--- проверим и закроем старые позиции, открытые более holdbars баров назад
ClosePositionsByBars(holdbars,slippage,EXPERT_MAGIC);
//--- проверим появление нового бара
if(isNewBar())
{
//--- проверим наличие сигнала
signal=CheckSignal();
}
}

```

```

        }

//--- если открыта неттинговая позиция, то сигнал пропускаем - ждем, пока она закроется
if(signal!=0 && PositionsTotal()>0 && (ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger
{
    signal=0;
    return; // выходим из обработчика события NewTick и не входим в рынок до появления
}

//--- для хеджинового счета каждая позиция живет и закрывается раздельно
if(signal!=0)
{
    //--- сигнал на покупку
    if(signal>0)
    {
        PrintFormat("%s: Есть сигнал на покупку! Revers=%s",__FUNCTION__,string(revers));
        if(Buy(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
    }

    //--- сигнал на продажу
    if(signal<0)
    {
        PrintFormat("%s: Есть сигнал на продажу! Revers=%s",__FUNCTION__,string(revers));
        if(Sell(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
    }
}

//--- конец функции OnTick
}

//-----+
//| Проверяет наличие торгового сигнала |
//-----+
int CheckSignal()
{
//--- 0 означает отсутствие сигнала
int res=0;

//--- получим значение ATR на предпоследнем завершенном баре (индекс бара равен 2)
double atr_value[1];
if(CopyBuffer(atr_handle,0,2,1,atr_value)!=-1)
{
    last_atr=atr_value[0];
//--- получим данные последнего закрытого бара в массив типа MqlRates
MqlRates bar[1];
if(CopyRates(_Symbol,_Period,1,1,bar)!=-1)
{
    //--- вычислим размер тела бара на последнем закрытом баре
    last_body=bar[0].close-bar[0].open;
//--- если тело последнего бара (с индексом 1) превышает предыдущее значение
    if(MathAbs(last_body)>kATR*last_atr)
        res=last_body>0?1:-1; // для растущей свечи положительное значение
}
}
}

```

```

        else
            PrintFormat("%s: Не удалось получить последний бар! Ошибка", __FUNCTION__, GetLastError());
    }
else
    PrintFormat("%s: Не удалось получить значение индикатора ATR! Ошибка", __FUNCTION__);
//--- если включен реверсивный режим торговли
res=reverse?-res:res; // если нужно, то развернем сигнал (вместо 1 вернем -1, а вместо -1 вернем 1)
//--- вернем значение торгового сигнала
return (res);
}

//+-----+
//| Возвращает true при появлении нового бара |+
//+-----+
bool isNewBar(const bool print_log=true)
{
    static datetime bartime=0; // храним время открытия текущего бара
//--- получим время открытия нулевого бара
datetime currbar_time=iTime(_Symbol,_Period,0);
//--- если время открытия изменилось, значит появился новый бар
if(bartime!=currbar_time)
{
    bartime=currbar_time;
    lastbar_timeopen=bartime;
//--- нужно ли выводить в лог информацию о времени открытия нового бара
if(print_log && !(MQLInfoInteger(MQL_OPTIMIZATION) || MQLInfoInteger(MQL_TESTER)))
{
//--- выведем сообщение о времени открытия нового бара
PrintFormat("%s: new bar on %s %s opened at %s", __FUNCTION__, _Symbol,
StringSubstr(EnumToString(_Period),7),
TimeToString(TimeCurrent(),TIME_SECONDS));
//--- получим данные о последнем тике
MqlTick last_tick;
if(!SymbolInfoTick(Symbol(),last_tick))
    Print("SymbolInfoTick() failed, error = ",GetLastError());
//--- выведем время последнего тика с точностью до миллисекунд
PrintFormat("Last tick was at %.%03d",
TimeToString(last_tick.time,TIME_SECONDS),last_tick.time_msc%1000);
}
//--- у нас есть новый бар
return (true);
}
//--- нового бара нет
return (false);
}

//+-----+
//| Покупка по рынку с заданным объемом |+
//+-----+
bool Buy(double volume, ulong deviation=10, ulong magicnumber=0)
{

```

```

//--- покупаем по рыночной цене
    return (MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber));
}
//+-----+
//| Продажа по рынку с заданным объемом |
//+-----+
bool Sell(double volume,ulong deviation=10,ulong magicnumber=0)
{
//--- продаем по рыночной цене
    return (MarketOrder(ORDER_TYPE_SELL,volume,deviation,magicnumber));
}
//+-----+
//| Закрытие позиций по времени удержания в барах |
//+-----+
void ClosePositionsByBars(int holdtimebars,ulong deviation=10,ulong magicnumber=0)
{
    int total=PositionsTotal(); // количество открытых позиций
//--- перебор всех открытых позиций
    for(int i=total-1; i>=0; i--)
    {
//--- параметры позиции
        ulong position_ticket=PositionGetTicket(i);
        string position_symbol=PositionGetString(POSITION_SYMBOL);
        ulong magic=PositionGetInteger(POSITION_MAGIC);
        datetime position_open=(datetime)PositionGetInteger(POSITION_TIME);
        int bars=iBarShift(_Symbol,PERIOD_CURRENT,position_open)+1;

//--- если позиция живет уже долго, а также MagicNumber и символ совпадают
        if(bars>holdtimebars && magic==magicnumber && position_symbol==_Symbol)
        {
            int digits=(int)SymbolInfoInteger(position_symbol,SYMBOL_DIGITS);
            double volume=PositionGetDouble(POSITION_VOLUME);
            ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
            string str_type=StringSubstr(EnumToString(type),14);
            StringToLower(str_type); // понижаем регистр текста для правильного форматиро
            PrintFormat("Закрываем позицию #%d %s %s %.2f",
                position_ticket,position_symbol,str_type,volume);
//--- установка типа ордера и отправки торгового запроса
            if(type==POSITION_TYPE_BUY)
                MarketOrder(ORDER_TYPE_SELL,volume,deviation,magicnumber,position_ticket);
            else
                MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber,position_ticket);
        }
    }
//+-----+
//| Подготовка и отправка торгового запроса |
//+-----+
bool MarketOrder(ENUM_ORDER_TYPE type,double volume,ulong slip,ulong magicnumber,ulong

```

```

{
//--- объявление и инициализация структур
MqlTradeRequest request={0};
MqlTradeResult result={0};
double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
if(type==ORDER_TYPE_BUY)
    price=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- параметры запроса
request.action =TRADE_ACTION_DEAL;                                // тип торговой операции
request.position =pos_ticket;                                         // тикет позиции, если за...
request.symbol =Symbol();                                            // символ
request.volume =volume;                                              // объем
request.type =type;                                                 // тип ордера
request.price =price;                                               // цена совершения сделки
request.deviation=slip;                                              // допустимое отклонение ...
request.magic =magicnumber;                                           // MagicNumber ордера
//--- отправка запроса
if(!OrderSend(request,result))
{
    //--- выведем информацию о неудаче
    PrintFormat("OrderSend %s %s %.2f at %.5f error %d",
                request.symbol,EnumToString(type),volume,request.price,GetLastError);
    return (false);
}
//--- сообщим об успешной операции
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
return (true);
}

```

#### Смотри также

[Функции обработки событий](#), [Выполнение программ](#), [События клиентского терминала](#), [OnTimer](#),  
[OnBookEvent](#), [OnChartEvent](#)

## OnCalculate

Вызывается в индикаторах при наступлении события [Calculate](#) для обработки изменений ценовых данных. Существуют два варианта функции, в пределах одного индикатора нельзя использовать оба варианта.

### Вычисление на основе массива данных

```
int OnCalculate(
    const int      rates_total,           // размер массива price[]
    const int      prev_calculated,     // количество обработанных баров на предыдущем
    const int      begin,                // номер индекса в массиве price[], с которого
    const double& price[]              // массив значений для расчета
);
```

### Вычисления на основе таймсерий текущего таймфрейма

```
int OnCalculate(
    const int      rates_total,           // размер входных таймсерий
    const int      prev_calculated,     // количество обработанных баров на предыдущем
    const datetime& time{},             // массив Time
    const double& open[],               // массив Open
    const double& high[],               // массив High
    const double& low[],                // массив Low
    const double& close[],              // массив Close
    const long&   tick_volume[],        // массив Tick Volume
    const long&   volume[],             // массив Real Volume
    const int&    spread[]              // массив Spread
);
```

### Параметры

*rates\_total*

[in] Размер массива `price[]` или входных таймсерий, доступных индикатору для расчета. Во втором варианте функции значение параметра соответствует количеству баров на графике, на котором он запущен.

*prev\_calculated*

[in] Содержит значение, которое вернула функция `OnCalculate()` на предыдущем вызове. Предназначено для пропуска в расчетах тех баров, которые не изменились с предыдущего запуска этой функции.

*begin*

[in] Значение индекса в массиве `price[]`, с которого начинаются значимые данные. Позволяет пропустить в расчетах отсутствующие или начальные данные, для которых нет корректных значений.

*price[]*

[in] Массив значений для проведения вычислений. В качестве массива `price[]` может быть передана одна из ценовых [таймсерий](#) либо рассчитанный буфер какого-либо индикатора. Тип

данных, которые были переданы на расчет, можно узнать с помощью предопределенной переменной [AppliedTo](#).

```
time{}  
[in] Массив со значениями времени открытия баров.  
  
open[]  
[in] Массив со значениями цен открытия.  
  
high[]  
[in] Массив со значениями максимальных цен.  
  
low[]  
[in] Массив со значениями минимальных цен.  
  
close[]  
[in] Массив со значениями цен закрытия.  
  
tick_volume[]  
[in] Массив со значениями тиковых объемов.  
  
volume[]  
[in] Массив со значениями торговых объемов.  
  
spread[]  
[in] Массив со значениями спреда для баров.
```

### Возвращаемое значение

Значение типа `int`, которое будет передано в качестве параметра `prev_calculated` при последующем вызове функции.

### Примечание

Если функция `OnCalculate()` возвращает нулевое значение, то в окне `DataWindow` клиентского терминала значения индикатора не показываются.

Если с момента последнего вызова функции `OnCalculate()` ценовые данные были изменены (была загружена более глубокая история или были заполнены пропуски истории), то значение входного параметра `prev_calculated` будет установлено в нулевое значение самим терминалом.

Чтобы определить направление индексации в массивах `time[]`, `open[]`, `high[]`, `low[]`, `close[]`, `tick_volume[]`, `volume[]` и `spread[]`, необходимо вызывать функцию [ArrayGetAsSeries\(\)](#). Чтобы не зависеть от умолчаний, необходимо безусловно вызывать функцию [ArraySetAsSeries\(\)](#) для тех массивов, с которыми предполагается работать.

При использовании первого варианта функции выбор необходимой таймсерией или индикатора в качестве массива `price[]` осуществляется пользователем на вкладке `Parameters` при запуске индикатора. Для этого необходимо указать нужный элемент в выпадающем списке поля ["Apply to"](#).

Для получения значений [пользовательского индикатора](#) из других MQL5-программ используется функция [iCustom\(\)](#), возвращающая хэндл индикатора для последующих операций. При этом также можно указать необходимый массив `price[]` или хэндл другого индикатора. Этот параметр должен передаваться последним в списке входных переменных пользовательского индикатора.

Необходимо использовать связь между значением, возвращаемым функцией OnCalculate(), и вторым входным параметром *prev\_calculated*. Параметр *prev\_calculated* при вызове функции содержит значение, которое вернула функция OnCalculate() на предыдущем вызове. Это позволяет реализовать экономные алгоритмы расчета пользовательского индикатора с тем, чтобы избежать повторных расчетов для тех баров, которые не изменились с предыдущего запуска этой функции.

### Пример индикатора

```
//+-----+
//|                                     OnCalculate_Sample.mq5 |
//|                                     Copyright 2018, MetaQuotes Software Corp. |
//|                                     https://www.mql5.com |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Пример вычисления индикатора Momentum"

//---- indicator settings
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots    1
#property indicator_type1    DRAW_LINE
#property indicator_color1   Blue
//---- входные параметры
input int MomentumPeriod=14; // Период для расчета
//---- индикаторный буфер
double    MomentumBuffer[];
//---- глобальная переменная для хранения периода расчетов
int       IntPeriod;
//+-----+
//| Custom indicator initialization function
//+-----+
void OnInit()
{
//--- проверим входной параметр
if(MomentumPeriod<0)
{
    IntPeriod=14;
    Print("параметр Период имеет неправильное значение. Для расчетов будет использован период 14");
}
else
    IntPeriod=MomentumPeriod;
//--- Буфера
    SetIndexBuffer(0,MomentumBuffer,INDICATOR_DATA);
//--- имя индикатора для показа в DataWindow и в подокне
    IndicatorSetString(INDICATOR_SHORTNAME, "Momentum"+(+string(IntPeriod)+""));
}
```

```

//--- установим номер бара, с которого будет идти отрисовка
PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,IntPeriod-1);
//--- установим 0.0 в качестве пустого значения, которое не отрисовывается
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
//--- с какой точностью показывать значения индикатора
IndicatorSetInteger(INDICATOR_DIGITS,2);
}

//-----+
//| Расчет индикатора Momentum |
//-----+
int OnCalculate(const int rates_total,      // размер массива price[]
                const int prev_calculated, // сколько баров обработано ранее
                const int begin,          // откуда начинаются значимые данные
                const double &price[])    // массив для значения для обработки
{
//--- стартовая позиция для вычислений
int StartCalcPosition=(IntPeriod-1)+begin;
//--- если недостаточно данных для расчета
if(rates_total<StartCalcPosition)
    return(0); // выходим с нулевым значением - индикатор не рассчитан
//--- correct draw begin
if(begin>0)
    PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,StartCalcPosition+(IntPeriod-1));
//--- начинаем расчеты, вычислим позицию начала
int pos=prev_calculated-1;
if(pos<StartCalcPosition)
    pos=begin+IntPeriod;
//--- основной цикл вычислений
for(int i=pos;i<rates_total && !IsStopped();i++)
    MomentumBuffer[i]=price[i]*100/price[i-IntPeriod];
//--- выполнение OnCalculate завершено. Вернем новое значение prev_calculated для пос
return(rates_total);
}

```

#### Смотри также

[ArrayGetAsSeries](#), [ArraySetAsSeries](#), [iCustom](#), [Функции обработки событий](#), [Выполнение программ](#), [События клиентского терминала](#), [Доступ к таймсериям и индикаторам](#)

## OnTimer

Вызывается в экспертах при наступлении события [Timer](#), генерируемого терминалом с заданным интервалом.

```
void OnTimer(void);
```

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

Событие Timer периодически генерируется клиентским терминалом для эксперта, который активизировал таймер при помощи функции [EventSetTimer\(\)](#). Обычно эта функция вызывается в функции [OnInit\(\)](#). При завершении работы эксперта необходимо уничтожить созданный таймер при помощи [EventKillTimer\(\)](#), которую обычно вызывают в функции [OnDeinit\(\)](#).

Каждый эксперт и каждый индикатор работают со своим таймером и получают события только от него. При завершении работы MQL5-программы таймер уничтожается принудительно, если он был создан, но не был отключен функцией [EventKillTimer\(\)](#).

Если необходимо получать события таймера чаще, чем один раз в секунду, используйте [EventSetMillisecondTimer\(\)](#) для создания таймера высокого разрешения.

В тестере стратегий используется минимальный интервал в 1000 миллисекунд. В общем случае при уменьшении периода таймера увеличивается время тестирования, так как возрастает количество вызовов обработчика событий таймера. При работе в режиме реального времени события таймера генерируются не чаще 1 раза в 10-16 миллисекунд, что связано с аппаратными ограничениями.

Для каждой программы может быть запущено не более одного таймера. Каждая MQL5-программа и каждый график имеют свою собственную очередь событий, куда складываются все вновь поступающие события. Если в очереди уже есть событие [Timer](#) либо это событие находится в состоянии обработки, то новое событие Timer в очередь mql5-программы не ставится.

### Пример эксперта с обработчиком OnTimer()

```
//+-----+
//|                                                 OnTimer_Sample.mq5 |
//|                                                 Copyright 2018, MetaQuotes Software Corp. |
//|                                                 https://www.mql5.com |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Пример использования таймера для вычисления времени торгового цикла"
#property description "Советник лучше всего запустить в конце торговой недели перед выходом"
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
```

```

{
//--- создадим таймер с периодом в 1 секунду
EventSetTimer(1);

//---
return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function |+
//+-----+
void OnDeinit(const int reason)
{
//--- уничтожим таймер при завершении работы
EventKillTimer();

}

//+-----+
//| Expert tick function |+
//+-----+
void OnTick()
{
//---


}

//+-----+
//| Timer function |+
//+-----+
void OnTimer()
{
//--- время первого вызова OnTimer()
static datetime start_time=TimeCurrent();
//--- время торгового сервера при первом вызове OnTimer();
static datetime start_tradeserver_time=0;
//--- вычисляемое время торгового сервера
static datetime calculated_server_time=0;
//--- локальное время на компьютере
datetime local_time=TimeLocal();
//--- текущее расчетное время торгового сервера
datetime trade_server_time=TimeTradeServer();
//--- если по какой-то причине время сервера неизвестно, то выходим досрочно
if(trade_server_time==0)
return;
//--- если стартовое значение торгового сервера еще не установлено
if(start_tradeserver_time==0)
{
start_tradeserver_time=trade_server_time;
//--- установим вычисляемое значение торгового сервера
Print(trade_server_time);
calculated_server_time=trade_server_time;
}
}

```

```
        }
    else
    {
        //--- увеличим время первого вызова OnTimer()
        if(start_tradeserver_time!=0)
            calculated_server_time=calculated_server_time+1;;
    }
//---

string com=StringFormat("Start time: %s\r\n",TimeToString(start_time));
com=com+StringFormat("Local time: %s\r\n",TimeToString(local_time));
com=com+StringFormat("TimeTradeServer time: %s\r\n",TimeToString(trade_server_time,
com=com+StringFormat(" EstimatedServer time: %s\r\n",TimeToString(calculated_server_time));
//--- выведем значения всех счетчиков на график
Comment(com);
}
```

#### Смотри также

[EventSetTimer](#), [EventSetMillisecondTimer](#), [EventKillTimer](#), [GetTickCount](#), [GetMicrosecondCount](#),  
[События клиентского терминала](#)

## OnTrade

Вызывается в экспертах при наступлении события [Trade](#). Функция предназначена для обработки изменений в списках ордеров, позиций и сделок.

```
void OnTrade(void);
```

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

`OnTrade()` вызывается только для экспертов, в индикаторах и скриптах она игнорируется, даже если добавить в них функцию с таким именем и типом.

При любом торговом действии (выставлении отложенного ордера, открытии/закрытии позиции, установке стопов, срабатывании отложенных ордеров и т.п.) соответствующим образом изменяется история ордеров и сделок и/или список позиций и текущих ордеров.

В момент обработки ордера торговый сервер посыпает терминалу сообщение о наступлении торгового события [Trade](#). Для получения из истории актуальных данных по ордерам и сделкам необходимо предварительно выполнить запрос торговой истории с помощью [HistorySelect\(\)](#).

Торговые события генерируются сервером в следующих случаях:

- изменение в действующих ордерах,
- изменения в позициях,
- изменения в сделках,
- изменения в торговой истории.

Каждое событие [Trade](#) может быть результатом одного или нескольких торговых запросов. Торговые запросы отправляются на сервер с помощью [OrderSend\(\)](#) или [OrderSendAsync\(\)](#). Каждый запрос может порождать несколько торговых событий. Нельзя полагаться на правило "Один запрос - Одно событие Trade", так как обработка запросов может происходить в несколько этапов и каждая операция может изменять состояние ордеров, позиций и торговой истории.

Обработчик [OnTrade\(\)](#) вызывается после соответствующих вызовов [OnTradeTransaction\(\)](#). В общем случае нет точного соотношения по количеству вызовов `OnTrade()` и `OnTradeTransaction()`. Один вызов `OnTrade()` соответствует одному или нескольким вызовам `OnTradeTransaction()`.

### Пример эксперта с обработчиком OnTrade()

```
//+-----+
//|                               OnTrade_Sample.mq5 |
//|                               Copyright 2018, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```

input      int days=7;                      // глубина торговой истории в днях
//--- зададим на глобальном уровне границы торговой истории
datetime    start;                         // дата начала торговой истории в кэше
datetime    end;                           // дата конца торговой истории в кэше
//--- глобальные счетчики
int         orders;                        // количество действующих ордеров
int         positions;                     // количество открытых позиций
int         deals;                         // количество сделок в кэше торговой истории
int         history_orders;                // количество ордеров в кэше торговой истории
bool        started=false;                  // флаг актуальности счетчиков

//+-----+
//| Expert initialization function          |
//+-----+
int OnInit()
{
//---
end=TimeCurrent();
start=end-days*PeriodSeconds(PERIOD_D1);
PrintFormat("Границы загружаемой торговой истории: начало - %s, конец - %s",
           TimeToString(start),TimeToString(end));
InitCounters();
//---
return(0);
}
//+-----+
//| инициализация счетчиков позиций, ордеров и сделок          |
//+-----+
void InitCounters()
{
ResetLastError();
//--- загрузим историю
bool selected=HistorySelect(start,end);
if(!selected)
{
PrintFormat("%s. Не удалось загрузить в кэш историю с %s по %s. Код ошибки: %d",
           __FUNCTION__,TimeToString(start),TimeToString(end),GetLastError());
return;
}
//--- получим текущие значения
orders=OrdersTotal();
positions=PositionsTotal();
deals=HistoryDealsTotal();
history_orders=HistoryOrdersTotal();
started=true;
Print("Счетчики ордеров, позиций и сделок успешно инициализированы");
}
//+-----+

```

```

//| Expert tick function
//+-----+
void OnTick()
{
    if(started) SimpleTradeProcessor();
    else InitCounters();
}
//+-----+
//| вызывается при поступлении события Trade
//+-----+
void OnTrade()
{
    if(started) SimpleTradeProcessor();
    else InitCounters();
}
//+-----+
//| пример обработки изменений в торговле и истории
//+-----+
void SimpleTradeProcessor()
{
    end=TimeCurrent();
    ResetLastError();
//--- загрузим в кэш программы торговую историю из указанного интервала
    bool selected=HistorySelect(start,end);
    if(!selected)
    {
        PrintFormat("%s. Не удалось загрузить в кэш историю с %s по %s. Код ошибки: %d",
                   __FUNCTION__, TimeToString(start), TimeToString(end), GetLastError());
        return;
    }
//--- получим текущие значения
    int curr_orders=OrdersTotal();
    int curr_positions=PositionsTotal();
    int curr_deals=HistoryDealsTotal();
    int curr_history_orders=HistoryOrdersTotal();
//--- проверим изменения в количестве действующих ордеров
    if(curr_orders!=orders)
    {
        //--- количество действующих ордеров изменилось
        PrintFormat("Изменилось количество ордеров. Было %d, стало %d",
                   orders,curr_orders);
        //--- обновим значение
        orders=curr_orders;
    }
//--- изменения в количестве открытых позиций
    if(curr_positions!=positions)
    {
        //--- количество открытых позиций изменилось
        PrintFormat("Изменилось количество позиций. Было %d, стало %d",
                   positions,curr_positions);
    }
}

```

```

        positions,curr_positions);
//--- обновим значение
positions=curr_positions;
}

//--- изменения в количестве сделок в кэше торговой истории
if(curr_deals!=deals)
{
    //--- количество сделок в кэше торговой истории изменилось
PrintFormat("Изменилось количество сделок. Было %d, стало %d",
            deals,curr_deals);
//--- обновим значение
deals=curr_deals;
}

//--- изменения в количестве исторических ордеров в кэше торговой истории
if(curr_history_orders!=history_orders)
{
    //--- количество исторических ордеров в кэше торговой истории изменилось
PrintFormat("Изменилось количество ордеров в истории. Было %d, стало %d",
            history_orders,curr_history_orders);
//--- обновим значение
history_orders=curr_history_orders;
}

//--- проверка на необходимость изменения границ торговой истории для запроса в кэш
CheckStartDateInTradeHistory();
}

//-----+
//|   изменения начальной даты для запроса торговой истории           |
//-----+
void CheckStartDateInTradeHistory()
{
//--- начальный интервал, если бы мы начали работу прямо сейчас
datetime curr_start=TimeCurrent()-days*PeriodSeconds(PERIOD_D1);
//--- убедимся, что граница начала торговой истории ушла не больше
//--- чем на 1 день от задуманной даты
if(curr_start-start>PeriodSeconds(PERIOD_D1))
{
    //--- придется подкорректировать дату начала загружаемой в кэш истории
start=curr_start;
PrintFormat("Новая граница начала загружаемой торговой истории: начало => %s",
           TimeToString(start));
//--- теперь заново загрузим торговую историю для обновленного интервала
HistorySelect(start,end);
//--- подкорректируем счетчики сделок и ордеров в истории для следующего сравнения
history_orders=HistoryOrdersTotal();
deals=HistoryDealsTotal();
}
}

//-----+
/* Пример вывода:

```

Границы загружаемой торговой истории: начало - 2018.07.16 18:11, конец - 2018.07.23

Счетчики ордеров, позиций и сделок успешно инициализированы

Изменилось количество ордеров. Было 0, стало 1

Изменилось количество ордеров. Было 1, стало 0

Изменилось количество позиций. Было 0, стало 1

Изменилось количество сделок. Было 0, стало 1

Изменилось количество ордеров в истории. Было 0, стало 1

\* /

#### Смотри также

[OrderSend](#), [OrderSendAsync](#), [OnTradeTransaction](#), [События клиентского терминала](#)

## OnTradeTransaction

Вызывается в экспертах при наступлении события [TradeTransaction](#). Функция предназначена для обработки результатов выполнения торгового запроса.

```
void OnTradeTransaction()
    const MqlTradeTransaction& trans,           // структура торговой транзакции
    const MqlTradeRequest& request,             // структура запроса
    const MqlTradeResult& result                // структура ответа
};
```

### Параметры

*trans*

[in] Переменная типа [MqlTradeTransaction](#) с описанием транзакции, проведенной на торговом счете.

*request*

[in] Переменная типа [MqlTradeRequest](#) с описанием торгового запроса, породившего транзакцию. Содержит значения только для транзакции типа [TRADE\\_TRANSACTION\\_REQUEST](#).

*result*

[in] Переменная типа [MqlTradeResult](#) с результатом выполнения торгового запроса, породившего транзакцию. Содержит значения только для транзакции типа [TRADE\\_TRANSACTION\\_REQUEST](#).

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

OnTradeTransaction() вызывается для обработки события [TradeTransaction](#), которое торговый сервер посыпает терминалу в следующих случаях:

- отправка торгового запроса из MQL5-программы при помощи функций [OrderSend\(\)](#)/[OrderSendAsync\(\)](#) и его последующее исполнение;
- отправка торгового запроса вручную через графический интерфейс и его последующее исполнение;
- срабатывания отложенных ордеров и стоп-ордеров на сервере;
- выполнения операций на стороне торгового сервера.

Информация о типе транзакции содержится в поле *type* переменной *trans*. Типы торговых транзакций описываются в перечислении [ENUM\\_TRADE\\_TRANSACTION\\_TYPE](#):

- **TRADE\_TRANSACTION\_ORDER\_ADD** - добавление нового действующего ордера
- **TRADE\_TRANSACTION\_ORDER\_UPDATE** - изменение действующего ордера
- **TRADE\_TRANSACTION\_ORDER\_DELETE** - удаление ордера из списка действующих
- **TRADE\_TRANSACTION\_DEAL\_ADD** - добавление сделки в историю
- **TRADE\_TRANSACTION\_DEAL\_UPDATE** - изменение сделки в истории
- **TRADE\_TRANSACTION\_DEAL\_DELETE** - удаление сделки из истории
- **TRADE\_TRANSACTION\_HISTORY\_ADD** - добавление ордера в историю в результате исполнения или отмены

- TRADE\_TRANSACTION\_HISTORY\_UPDATE - изменение ордера, находящегося в истории ордеров
- TRADE\_TRANSACTION\_HISTORY\_DELETE - удаление ордера из истории ордеров
- TRADE\_TRANSACTION\_POSITION - изменение позиции, не связанное с исполнением сделки
- TRADE\_TRANSACTION\_REQUEST - уведомление о том, что торговый запрос обработан сервером и результат его обработки получен.

При обработке транзакций типа TRADE\_TRANSACTION\_REQUEST для получения дополнительной информации необходимо анализировать второй и третий параметры функции OnTradeTransaction() - *request* и *result*.

Отправка торгового запроса на покупку приводит к цепи торговых транзакций, которые совершаются на торговом счете: 1) запрос принимается на обработку, 2) далее для счета создается соответствующий ордер на покупку, 3) затем происходит исполнение ордера, 4) удаление исполненного ордера из списка действующих, 5) добавление в историю ордеров, 6) далее добавляется соответствующая сделка в историю и 7) создается новая позиция. Все эти действия являются [торговыми транзакциями](#). Приход каждой такой транзакции в терминал является событием [TradeTransaction](#). При этом очередность поступления этих транзакций в терминал не гарантирована, поэтому нельзя свой торговый алгоритм строить на ожидании поступления одних торговых транзакций после прихода других.

Во время обработки торговых транзакций экспертом при помощи обработчика OnTradeTransaction(), терминал продолжает обрабатывать вновь поступающие торговые транзакции. Таким образом, состояние торгового счета может измениться уже в процессе работы OnTradeTransaction(). Например, пока MQL5-программа обрабатывает событие добавления нового ордера, он может быть исполнен, удален из списка открытых и перемещен в историю. В дальнейшем программа будет уведомлена о всех этих событиях.

Длина очереди транзакций составляет 1024 элемента. Если OnTradeTransaction() будет обрабатывать очередную транзакцию слишком долго, старые транзакции в очереди могут быть вытеснены более новыми.

Обработчик [OnTrade\(\)](#) вызывается после соответствующих вызовов OnTradeTransaction(). В общем случае нет точного соотношения по количеству вызовов OnTrade() и OnTradeTransaction(). Один вызов OnTrade() соответствует одному или нескольким вызовам OnTradeTransaction.

Каждое событие [Trade](#) может быть результатом одного или нескольких торговых запросов. Торговые запросы отправляются на сервер с помощью [OrderSend\(\)](#) или [OrderSendAsync\(\)](#). Каждый запрос может порождать несколько торговых событий. Нельзя полагаться на правило "Один запрос - Одно событие Trade", так как обработка запросов может происходить в несколько этапов и каждая операция может изменять состояние ордеров, позиций и торговой истории.

#### Пример эксперта с обработчиком OnTradeTransaction()

```
//+-----+
//|                               OnTradeTransaction_Sample.mq5  |
//|                               Copyright 2018, MetaQuotes Software Corp.  |
//|                               https://www.mql5.com  |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```

#property description "Пример слушателя событий TradeTransaction"
//+-----+
//| Expert initialization function | 
//+-----+
int OnInit()
{
//---
PrintFormat("LAST PING=% .f ms",
TerminalInfoInteger(TERMINAL_PING_LAST)/1000.);
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Expert tick function | 
//+-----+
void OnTick()
{
//---

}

//+-----+
//| TradeTransaction function | 
//+-----+
void OnTradeTransaction(const MqlTradeTransaction &trans,
const MqlTradeRequest &request,
const MqlTradeResult &result)
{
//---
static int counter=0; // счетчик вызовов OnTradeTransaction()
static uint lasttime=0; // время последнего вызова OnTradeTransaction()
//---
uint time=GetTickCount();
//--- если последняя транзакция была больше 1 секунды назад
if(time-lasttime>1000)
{
counter=0; // значит, это новая торговая операция и можно сбросить счетчик
if(IS_DEBUG_MODE)
Print(" Новая торговая операция");
}
lasttime=time;
counter++;
Print(counter,". ",__FUNCTION__);
//--- результат выполнения торгового запроса
ulong lastOrderID =trans.order;
ENUM_ORDER_TYPE lastOrderType =trans.order_type;
ENUM_ORDER_STATE lastOrderState=trans.order_state;
//--- имя символа, по которому произошла транзакция
string trans_symbol=trans.symbol;
//--- тип транзакции

```

```

ENUM_TRADE_TRANSACTION_TYPE trans_type=trans.type;
switch(trans.type)
{
    case TRADE_TRANSACTION_POSITION: // изменение позиции
    {
        ulong pos_ID=trans.position;
        PrintFormat("MqlTradeTransaction: Position # %d %s modified: SL=%f TP=%f",
                    pos_ID,trans_symbol,trans.price_sl,trans.price_tp);
    }
    break;
    case TRADE_TRANSACTION_REQUEST: // отправка торгового запроса
        PrintFormat("MqlTradeTransaction: TRADE_TRANSACTION_REQUEST");
        break;
    case TRADE_TRANSACTION_DEAL_ADD: // добавление сделки
    {
        ulong lastDealID =trans.deal;
        ENUM DEAL_TYPE lastDealType =trans.deal_type;
        double lastDealVolume=trans.volume;
        //--- идентификатор сделки во внешней системе - тикет, присваиваемый биржей
        string Exchange_ticket="";
        if(HistoryDealSelect(lastDealID))
            Exchange_ticket=HistoryDealGetString(lastDealID,DEAL_EXTERNAL_ID);
        if(Exchange_ticket!="")
            Exchange_ticket=StringFormat("(Exchange deal=%s)",Exchange_ticket);

        PrintFormat("MqlTradeTransaction: %s deal # %d %s %s %.2f lot %s",EnumToString(
                    lastDealID,EnumToString(lastDealType),trans_symbol,lastDealVolume));
    }
    break;
    case TRADE_TRANSACTION_HISTORY_ADD: // добавление ордера в историю
    {
        //--- идентификатор ордера во внешней системе - тикет, присваиваемый биржей
        string Exchange_ticket="";
        if(lastOrderState==ORDER_STATE_FILLED)
        {
            if(HistoryOrderSelect(lastOrderID))
                Exchange_ticket=HistoryOrderGetString(lastOrderID,ORDER_EXTERNAL_ID);
            if(Exchange_ticket!="")
                Exchange_ticket=StringFormat("(Exchange ticket=%s)",Exchange_ticket);
        }
        PrintFormat("MqlTradeTransaction: %s order # %d %s %s %s",EnumToString(
                    lastOrderID,EnumToString(lastOrderType),trans_symbol,EnumToString(
                    lastOrderState)));
    }
    break;
    default: // прочие транзакции
    {
        //--- идентификатор ордера во внешней системе - тикет, присваиваемый Московской
        string Exchange_ticket="";
        if(lastOrderState==ORDER_STATE_PLACED)
            Exchange_ticket=StringFormat("(Exchange ticket=%s)",Exchange_ticket);
    }
}

```

```

    {
        if(OrderSelect(lastOrderID))
            Exchange_ticket=OrderGetString(ORDER_EXTERNAL_ID);
        if(Exchange_ticket!="")
            Exchange_ticket=StringFormat("Exchange ticket=%s",Exchange_ticket);
    }
    PrintFormat("MqlTradeTransaction: %s order #%d %s %s %s",EnumToString(transaction),
                lastOrderID,EnumToString(lastOrderType),EnumToString(lastOrderStatus));
}
break;
}

//--- тикет ордера
ulong orderID_result=result.order;
string retcode_result=GetRetcodeID(result.retcode);
if(orderID_result!=0)
    PrintFormat("MqlTradeResult: order #%d retcode=%s ",orderID_result,retcode_result);
//---
}

//+-----+
//| переводит числовые коды ответов в строковые мнемокоды |
//+-----+
string GetRetcodeID(int retcode)
{
    switch(retcode)
    {
        case 10004: return("TRADE_RETCODE_QUOTE"); break;
        case 10006: return("TRADE_RETCODE_REJECT"); break;
        case 10007: return("TRADE_RETCODE_CANCEL"); break;
        case 10008: return("TRADE_RETCODE_PLACED"); break;
        case 10009: return("TRADE_RETCODE_DONE"); break;
        case 10010: return("TRADE_RETCODE_DONE_PARTIAL"); break;
        case 10011: return("TRADE_RETCODE_ERROR"); break;
        case 10012: return("TRADE_RETCODE_TIMEOUT"); break;
        case 10013: return("TRADE_RETCODE_INVALID"); break;
        case 10014: return("TRADE_RETCODE_INVALID_VOLUME"); break;
        case 10015: return("TRADE_RETCODE_INVALID_PRICE"); break;
        case 10016: return("TRADE_RETCODE_INVALID_STOPS"); break;
        case 10017: return("TRADE_RETCODE_TRADE_DISABLED"); break;
        case 10018: return("TRADE_RETCODE_MARKET_CLOSED"); break;
        case 10019: return("TRADE_RETCODE_NO_MONEY"); break;
        case 10020: return("TRADE_RETCODE_PRICE_CHANGED"); break;
        case 10021: return("TRADE_RETCODE_PRICE_OFF"); break;
        case 10022: return("TRADE_RETCODE_INVALID_EXPIRATION"); break;
        case 10023: return("TRADE_RETCODE_ORDER_CHANGED"); break;
        case 10024: return("TRADE_RETCODE_TOO_MANY_REQUESTS"); break;
        case 10025: return("TRADE_RETCODE_NO_CHANGES"); break;
        case 10026: return("TRADE_RETCODE_SERVER_DISABLES_AT"); break;
        case 10027: return("TRADE_RETCODE_CLIENT_DISABLES_AT"); break;
        case 10028: return("TRADE_RETCODE_LOCKED"); break;
    }
}

```

```
case 10029: return("TRADE_RETCODE_FROZEN"); break;
case 10030: return("TRADE_RETCODE_INVALID_FILL"); break;
case 10031: return("TRADE_RETCODE_CONNECTION"); break;
case 10032: return("TRADE_RETCODE_ONLY_REAL"); break;
case 10033: return("TRADE_RETCODE_LIMIT_ORDERS"); break;
case 10034: return("TRADE_RETCODE_LIMIT_VOLUME"); break;
case 10035: return("TRADE_RETCODE_INVALID_ORDER"); break;
case 10036: return("TRADE_RETCODE_POSITION_CLOSED"); break;
default:
    return("TRADE_RETCODE_UNKNOWN="+IntegerToString(retcode));
    break;
}
//---
}
```

#### Смотри также

[OrderSend](#), [OrderSendAsync](#), [OnTradeTransaction](#), [Структура торгового запроса](#), [Структура торговой транзакции](#), [Типы торговых транзакций](#), [Типы торговых операций](#), [События клиентского терминала](#)

## OnBookEvent

Вызывается в индикаторах и экспертах при наступлении события [BookEvent](#). Функция предназначена для обработки изменений стакана цен (Depth of Market).

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

### Параметры

*symbol*

[in] Имя финансового инструмента, по которому пришло событие [BookEvent](#)

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

Чтобы получать события BookEvent по любому символу, достаточно предварительно подписаться на получение этих событий для этого символа с помощью функции [MarketBookAdd\(\)](#). Для того чтобы отписаться от получения события BookEvent по конкретному символу, необходимо вызывать функцию [MarketBookRelease\(\)](#).

Событие BookEvent является широковещательным в пределах графика. Это означает, что достаточно одному приложению на графике подписаться на получение события BookEvent с помощью функции MarketBookAdd, как все остальные индикаторы и эксперты, запущенные на этом графике и имеющие обработчик OnBookEvent(), будут получать это событие. Поэтому необходимо анализировать имя символа, которое передается в обработчик OnBookEvent() в качестве параметра *symbol*.

Для всех приложений, запущенных на одном графике, ведутся отдельные счетчики на получение событий BookEvent в разрезе символов. Это означает, что на каждом графике может быть несколько подписок на разные символы, и для каждого символа ведется свой собственный счетчик. Подписка и отписка событий BookEvent изменяют счетчик подписок только указанных символов, но при этом только в пределах одного графика. Это означает, что на двух соседних графиках могут быть подписки на события BookEvent на один и тот же символ, но с разными значениями счетчиков подписок.

Начальное значение счетчика подписок равно нулю. При каждом вызове [MarketBookAdd\(\)](#) счетчик подписок для указанного символа на данном графике увеличивается на единицу (символ графика и символ в MarketBookAdd() не обязаны совпадать). При вызове [MarketBookRelease\(\)](#) счетчик подписок на указанный символ в пределах графика уменьшается на единицу. Трансляция событий BookEvent по любому символу в пределах графика продолжается до тех пор, пока счетчик подписок по данному символу не станет равным нулю. Поэтому важно, чтобы каждая MQL5-программа, которая содержит вызовы [MarketBookAdd\(\)](#), при завершении своей работы правильно отписывалась от получения событий по каждому использованному символу с помощью [MarketBookRelease\(\)](#). Для этого достаточно, чтобы количество вызовов [MarketBookAdd\(\)](#) и [MarketBookRelease\(\)](#) по каждому вызову было четным за всё время жизни MQL5-программы. Использование флагов или собственных счетчиков подписок внутри программы позволяет безопасно работать с событиями BookEvent и предотвращает отключение подписок на получение этого события в чужих программах в пределах одного графика.

События [BookEvent](#) никогда не пропускаются и всегда ставятся в очередь, даже если в данный момент еще не закончена обработка предыдущего события BookEvent. При этом необходимо иметь в виду, что события BookEvent доставляются сами по себе и не несут с собой состояния стакана заявок. Это означает, что вызов [MarketBookGet\(\)](#) из обработчика OnBookEvent() позволяет получить текущее актуальное состояние стакана на момент вызова, а не то состояние стакана, которое вызвало отправку события [BookEvent](#). Для гарантированного получения всех уникальных состояний стакана функция OnBookEvent() должна быть максимально быстрой.

### Пример

```
//+-----+
//|                               OnBookEvent_Sample.mq5  |
//|                               Copyright 2018, MetaQuotes Software Corp.  |
//|                               https://www.mql5.com  |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com/ru/articles/2635"
#property version   "1.00"
#property description "Пример замера скорости обновления стакана с помощью OnBookEvent"
#property description "Код взят из статьи https://www.mql5.com/ru/articles/2635"
//--- входные параметры
input ulong ExtCollectTime    =30; // время теста в секундах
input ulong ExtSkipFirstTicks=10; // количество пропускаемых тиков на старте
//--- флаг наличия подписки на получение событий BookEvent
bool book_subscribed=false;
//--- массив для приема заявок из стакана
MqlBookInfo book[];
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- покажем старт
Comment(StringFormat("Ждем поступления первых %I64u тиков",ExtSkipFirstTicks));
PrintFormat("Ждем поступления первых %I64u тиков",ExtSkipFirstTicks);
//--- включим трансляцию стакана
if(MarketBookAdd(_Symbol))
{
book_subscribed=true;
PrintFormat("%s: Функция MarketBookAdd(%s) вернула true",__FUNCTION__,__Symbol);
}
else
PrintFormat("%s: Функция MarketBookAdd(%s) вернула false! GetLastError()=%d",__FUNCTION__,__Symbol);
//--- успешная инициализация
return(INIT_SUCCEEDED);
}
//+-----+
//| Deinitialize expert
//+-----+
```

```

//+-----+
void OnDeinit(const int reason)
{
//--- выводим код причины deinициализации
Print(__FUNCTION__,": Код причины deinициализации = ",reason);
//--- отменим свою подписку на получение событий стакана
if(book_subscribed)
{
    if(!MarketBookRelease(_Symbol))
        PrintFormat("%s: MarketBookRelease(%s) вернула false! GetLastError()=%d",_Sy
    else
        book_subscribed=false;
}
//---
}

//+-----+
//| BookEvent function
//+-----+
void OnBookEvent(const string &symbol)
{
    static ulong starttime=0;           // время начала теста
    static ulong tickcounter=0;         // счетчик обновлений стакана цен
//--- работаем с событиями стакана только в том случае, если мы сами на него подписали
    if(!book_subscribed)
        return;
//--- считаем обновления только по своему символу
    if(symbol!=_Symbol)
        return;
//--- пропускаем первые тики для первичной очистки очереди и разогрева
    tickcounter++;
    if(tickcounter<ExtSkipFirstTicks)
        return;
//--- запомним время старта
    if(tickcounter==ExtSkipFirstTicks)
        starttime=GetMicrosecondCount();
//--- запросим данные стакана
    MarketBookGet(symbol,book);
//--- когда надо остановиться?
    ulong endtime=GetMicrosecondCount()-starttime;
    ulong ticks =1+tickcounter-ExtSkipFirstTicks;
// сколько прошло времени в микросекундах с начала теста?
    if(endtime>ExtCollectTime*1000*1000)
    {
        PrintFormat("%I64u ticks for %.1f seconds: %.1f ticks/sec ",ticks,endtime/1000.0
        ExpertRemove();
        return;
    }
//--- вывод счетчиков в поле комментария
}

```

```
if(endtime>0)
    Comment(StringFormat("%I64u ticks for %.1f seconds: %.1f ticks/sec ",ticks,endtime));
}
```

#### Смотри также

[MarketBookAdd](#), [MarketBookRelease](#), [MarketBookGet](#), [OnTrade](#), [OnTradeTransaction](#), [OnTick](#),  
[Функции обработки событий](#), [Выполнение программ](#), [События клиентского терминала](#)

## OnChartEvent

Вызывается в экспертах и индикаторах при наступлении события [ChartEvent](#). Функция предназначена для обработки изменений графика, вызванных действиями пользователя или работой MQL5-программ.

```
void OnChartEvent()
    const int      id,          // идентификатор события
    const long&    lparam,       // параметр события типа long
    const double&  dparam,      // параметр события типа double
    const string&  sparam       // параметр события типа string
);
```

### Параметры

*id*

[in] Идентификатор события из перечисления [ENUM\\_CHART\\_EVENT](#).

*lparam*

[in] Параметр события типа [long](#)

*dparam*

[in] Параметр события типа [double](#)

*sparam*

[in] Параметр события типа [string](#)

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

Существуют 11 видов событий, которые можно обрабатывать с помощью предопределенной функции OnChartEvent(). Для пользовательских событий предусмотрено 65535 идентификаторов в диапазоне от CHARTEVENT\_CUSTOM до CHARTEVENT\_CUSTOM\_LAST включительно. Для генерации пользовательского события необходимо использовать функцию [EventChartCustom\(\)](#).

Краткое описание событий из перечисления [ENUM\\_CHART\\_EVENT](#):

- CHARTEVENT\_KEYDOWN – нажатие клавиатуры, когда окно графика находится в фокусе;
- CHARTEVENT\_MOUSE\_MOVE – перемещение мыши и нажатия кнопок мыши (если для графика установлено свойство [CHART\\_EVENT\\_MOUSE\\_MOVE=true](#));
- CHARTEVENT\_OBJECT\_CREATE – создание [графического объекта](#) (если для графика установлено свойство [CHART\\_EVENT\\_OBJECT\\_CREATE=true](#));
- CHARTEVENT\_OBJECT\_CHANGE – изменение свойств объекта через диалог свойств;
- CHARTEVENT\_OBJECT\_DELETE – удаление графического объекта (если для графика установлено свойство [CHART\\_EVENT\\_OBJECT\\_DELETE=true](#));
- CHARTEVENT\_CLICK – щелчок мыши на графике;
- CHARTEVENT\_OBJECT\_CLICK – щелчок мыши на графическом объекте, принадлежащем графику;
- CHARTEVENT\_OBJECT\_DRAG – перемещение графического объекта при помощи мыши;

- CHARTEVENT\_OBJECT\_ENDEDIT – окончание редактирования текста в поле ввода графического объекта Edit ([OBJ\\_EDIT](#));
- CHARTEVENT\_CHART\_CHANGE – изменения графика;
- CHARTEVENT\_CUSTOM+n – идентификатор пользовательского события, где n находится в диапазоне от 0 до 65535. CHARTEVENT\_CUSTOM\_LAST содержит последний допустимый идентификатор пользовательского события (CHARTEVENT\_CUSTOM+65535).

Все [MQL5-программы](#) работают в потоках, отличных от главного потока приложения. Главный поток терминала отвечает за обработку всех системных сообщений Windows, и в результате этой обработки в свою очередь порождает сообщения Windows для своего же приложения. Например, перемещение мышки на графике (событие WM\_MOUSE\_MOVE) порождает несколько системных сообщений для последующей отрисовки окна приложения, а также посыпает внутренние сообщения экспертам и индикаторам, запущенным на этом графике. При этом может возникнуть ситуация, что главный поток приложения ещё не успел обработать системное сообщение WM\_PAINT (и поэтому ещё не отрисовал изменённый график), а эксперт или индикатор уже получили событие о перемещении курсора мыши. Тогда в этом случае свойство графика CHART\_FIRST\_VISIBLE\_BAR будет изменено только после отрисовки графика.

Для каждого типа события входные параметры функции OnChartEvent() имеют определенные значения, которые необходимы для обработки этого события. В таблице перечислены события и значения, которые передаются через параметры.

Событие	Значение параметра id	Значение параметра lparam	Значение параметра dparam	Значение параметра sparam
Событие нажатия клавиатуры	CHARTEVENT_KEYDOWN	код нажатой клавиши	Количество нажатий клавиши, сгенерированных за время ее удержания в нажатом состоянии	Строковое значение битовой маски, описывающее статус кнопок клавиатуры
События мышки (если для графика установлено свойство <a href="#">CHART_EVENT_MOUSE_MOVE=true</a> )	CHARTEVENT_MOUSE_MOVE	X-координата	Y-координата	Строковое значение битовой маски, описывающее статус кнопок мыши
Событие колесика мышки (если для графика установлено свойство <a href="#">CHART_EVENT_MOUSE_WHEEL=true</a> )	CHARTEVENT_MOUSE_WHEEL	Флаги состояний клавиш и кнопок мышки, координаты X и Y курсора. Описание дано в <a href="#">примере</a>	Значение Delta прокрутки колесика мышки	–

Событие создания графического объекта (если для графика установлено свойство <u><a href="#">CHART_EVENT_OBJECT_CREATE</a></u> =true)	CHARTEVENT_OBJECT_CREATE	—	—	Имя созданного графического объекта
Событие изменения свойств объекта через диалог свойств	CHARTEVENT_OBJECT_CHANGE	—	—	Имя измененного графического объекта
Событие удаления графического объекта (если для графика установлено свойство <u><a href="#">CHART_EVENT_OBJECT_DELETE</a></u> =true)	CHARTEVENT_OBJECT_DELETE	—	—	Имя удаленного графического объекта
Событие щелчка мышки на графике	CHARTEVENT_CLICK	X-координата	Y-координата	—
Событие щелчка мышки на графическом объекте	CHARTEVENT_OBJECT_CLICK	X-координата	Y-координата	Имя графического объекта, на котором произошло событие
Событие перемещения графического объекта при помощи мышки	CHARTEVENT_OBJECT_DRAG	—	—	Имя перемещенного графического объекта
Событие окончания редактирования текста в поле ввода графического объекта "Поле ввода"	CHARTEVENT_OBJECT_ENDEDIT	—	—	Имя графического объекта "Поле ввода", в котором завершилось редактирование текста

Событие изменения размеров графика или изменения свойств графика через диалог свойств	CHARTEVENT_C HART_CHANGE	—	—	—
Пользовательское событие с номером N	CHARTEVENT_C USTOM+N	Значение, заданное функцией <a href="#">EventChartCustom()</a>	Значение, заданное функцией <a href="#">EventChartCustom()</a>	Значение, заданное функцией <a href="#">EventChartCustom()</a>

Пример слушателя событий графика:

```

//+-----+
//| OnChartEvent_Sample.mq5 |
//| Copyright 2018, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Пример слушателя событий графика и генератора пользовательских
//--- идентификаторы служебных клавиш
#define KEY_NUMPAD_5      12
#define KEY_LEFT          37
#define KEY_UP            38
#define KEY_RIGHT         39
#define KEY_DOWN          40
#define KEY_NUMLOCK_DOWN  98
#define KEY_NUMLOCK_LEFT  100
#define KEY_NUMLOCK_5     101
#define KEY_NUMLOCK_RIGHT 102
#define KEY_NUMLOCK_UP    104
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- выведем значение константы CHARTEVENT_CUSTOM
Print("CHARTEVENT_CUSTOM=",CHARTEVENT_CUSTOM);
//---
Print("Запущен эксперт с именем ",MQLInfoString(MQL5_PROGRAM_NAME));
//--- установка флага получения событий создания объектов графика
ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_CREATE,true);
//--- установка флага получения событий удаления объектов графика
ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_DELETE,true);
}

```

```

//--- включение сообщений о прокрутке колесика мышки
ChartSetInteger(0,CHART_EVENT_MOUSE_WHEEL,1);
//--- принудительное обновление свойств графика гарантирует готовность к обработке событий
ChartRedraw();
//---

    return(INIT_SUCCEEDED);
}

//+-----+
//| Expert tick function
//+-----+
void OnTick()
{
//--- счетчик тиков для генерации пользовательского события
static int tick_counter=0;
//--- будем делить накопленные тики на это число
int simple_number=113;
//---

    tick_counter++;
//--- отправляем пользовательское событие, если счетчик тиков кратен simple_number
if(tick_counter%simple_number==0)
{
//--- сформируем идентификатор пользовательского события в диапазоне от 0 до 65535
ushort custom_event_id=ushort(tick_counter%65535);
//--- отправим пользовательское событие с заполнением параметров
EventChartCustom(ChartID(),custom_event_id,tick_counter,SymbolInfoDouble(Symbol,
//--- сделаем вывод в лог для изучения и анализа результатов примера
Print(__FUNCTION__,": Отправлено пользовательское событие ID=",custom_event_id));
}
//---

}

//+-----+
//| ChartEvent function
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
//--- нажатие кнопки на клавиатуре
if(id==CHARTEVENT_KEYDOWN)
{
switch((int)lparam)
{
    case KEY_NUMLOCK_LEFT: Print("Нажата KEY_NUMLOCK_LEFT"); break;
    case KEY_LEFT:          Print("Нажата KEY_LEFT"); break;
    case KEY_NUMLOCK_UP:    Print("Нажата KEY_NUMLOCK_UP"); break;
    case KEY_UP:            Print("Нажата KEY_UP"); break;
    case KEY_NUMLOCK_RIGHT: Print("Нажата KEY_NUMLOCK_RIGHT"); break;
    case KEY_RIGHT:         Print("Нажата KEY_RIGHT"); break;
}
}
}

```

```

        case KEY_NUMLOCK_DOWN: Print("Нажата KEY_NUMLOCK_DOWN"); break;
        case KEY_DOWN: Print("Нажата KEY_DOWN"); break;
        case KEY_NUMPAD_5: Print("Нажата KEY_NUMPAD_5"); break;
        case KEY_NUMLOCK_5: Print("Нажата KEY_NUMLOCK_5"); break;
        default: Print("Нажата какая-то неперечисленная клавиша");
    }
}

//--- нажатие левой кнопкой мышки на графике
if(id==CHARTEVENT_CLICK)
    Print("Координаты щелчка мышки на графике: x = ",lparam," y = ",dparam);
//--- нажатие мышкой на графическом объекте
if(id==CHARTEVENT_OBJECT_CLICK)
    Print("Нажатие кнопки мышки на объекте с именем '"+sparam+"'");
//--- удален объект
if(id==CHARTEVENT_OBJECT_DELETE)
    Print("Удален объект с именем ",sparam);
//--- создан объект
if(id==CHARTEVENT_OBJECT_CREATE)
    Print("Создан объект с именем ",sparam);
//--- изменен объект
if(id==CHARTEVENT_OBJECT_CHANGE)
    Print("Изменен объект с именем ",sparam);
//--- перемещен объект или изменены координаты точек привязки
if(id==CHARTEVENT_OBJECT_DRAG)
    Print("Изменение точек привязки объекта с именем ",sparam);
//--- изменен текст в поле ввода графического объекта Edit
if(id==CHARTEVENT_OBJECT_ENDEDIT)
    Print("Изменен текст в объекте Edit ",sparam," id=",id);
//--- события перемещения мышки
if(id==CHARTEVENT_MOUSE_MOVE)
    Comment("POINT: ",(int)lparam,",", (int)dparam,"\\n",MouseState((uint)sparam));
if(id==CHARTEVENT_MOUSE_WHEEL)
{
    //--- разберем состояние кнопок и колесика мышки для этого события
    int flg_keys = (int)(lparam>>32);           // флаг состояний клавиш Ctrl, Shift
    int x_cursor = (int)(short)lparam;             // X-координата, в которой произошло
    int y_cursor = (int)(short)(lparam>>16);     // Y-координата, в которой произошло
    int delta    = (int)dparam;                   // суммарное значение прокрутки колеса
    //--- обрабатываем флаг
    string str_keys="";
    if((flg_keys&0x0001)!=0)
        str_keys+="LMOUSE ";
    if((flg_keys&0x0002)!=0)
        str_keys+="RMOUSE ";
    if((flg_keys&0x0004)!=0)
        str_keys+="SHIFT ";
    if((flg_keys&0x0008)!=0)
        str_keys+="CTRL ";
    if((flg_keys&0x0010)!=0)

```

```

        str_keys+="MMOUSE ";
        if((flg_keys&0x0020)!=0)
            str_keys+="X1MOUSE ";
        if((flg_keys&0x0040)!=0)
            str_keys+="X2MOUSE ";

        if(str_keys!="")
            str_keys=", keys='"+StringSubstr(str_keys,0,StringLen(str_keys)-1)+"'";
        PrintFormat ("%s: X=%d, Y=%d, delta=%d%s",EnumToString(CHARTEVENT_MOUSE_WHEEL),x_
        }

//--- изменение размеров графика или изменение свойств графика через диалог свойств
if(id==CHARTEVENT_CHART_CHANGE)
    Print ("Изменение размеров или свойств графика");
//--- пользовательское событие
if(id>CHARTEVENT_CUSTOM)
    PrintFormat ("Пользовательское событие ID=%d, lparam=%d, dparam=%G, sparam=%s",id_
    }

//+-----+
//| MouseState |
//+-----+
string MouseState(uint state)
{
    string res;
    res+="\nML: " +(((state& 1)== 1)?"DN":"UP"); // mouse left
    res+="\nMR: " +(((state& 2)== 2)?"DN":"UP"); // mouse right
    res+="\nMM: " +(((state&16)==16)?"DN":"UP"); // mouse middle
    res+="\nMX: " +(((state&32)==32)?"DN":"UP"); // mouse first X key
    res+="\nMY: " +(((state&64)==64)?"DN":"UP"); // mouse second X key
    res+="\nSHIFT: "+(((state& 4)== 4)?"DN":"UP"); // shift key
    res+="\nCTRL: " +(((state& 8)== 8)?"DN":"UP"); // control key
    return(res);
}

```

#### Смотри также

[EventChartCustom](#), [Типы событий графика](#), [Функции обработки событий](#), [Выполнение программ](#), [События клиентского терминала](#)

## OnTester

Вызывается в экспертах при наступлении события [Tester](#) для выполнения необходимых действий по окончании тестирования.

```
double OnTester(void);
```

### Возвращаемое значение

Значение пользовательского критерия оптимизации для оценки результатов тестирования.

### Примечание

Функция `OnTester()` может быть использована только в экспертах при тестировании и предназначена в первую очередь для расчета некоторого значения, используемого в качестве критерия "Custom max" при оптимизации входных параметров.

При генетической оптимизации сортировка результатов в пределах одного поколения производится по убыванию. Это означает, что лучшими с точки зрения критерия оптимизации считаются результаты с наибольшим значением. Худшие значения при такой сортировке помещаются в конец и впоследствии отбрасываются и не принимают участия в формировании следующего поколения.

Таким образом, с помощью функции `OnTester()` можно не только создавать и сохранять собственные отчеты результатов тестирования, но и управлять ходом оптимизации для поиска наилучших параметров торговой стратегии.

**Пример** расчета пользовательского критерия оптимизации. Идея заключается в вычислении линейной регрессии графика баланса и описана в статье [Оптимизируем стратегию по графику баланса и сравниваем результаты с критерием "Balance + max Sharpe Ratio"](#)

```
//+-----+
//|                               OnTester_Sample.mq5 |
//|           Copyright 2018, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Пример советника с обработчиком OnTester()"
#property description "В качестве пользовательского критерия оптимизации "
#property description "возвращается коэффициент линейной регрессии графика баланса,"
#property description "деленный на среднеквадратичную ошибку отклонения"
//--- подключим класс торговых операций
#include <Trade\Trade.mqh>
//--- входные параметры эксперта
input double Lots          = 0.1;        // Объем
input int    Slippage       = 10;         // Допустимое проскальзывание
input int    MovingPeriod   = 80;         // Период скользящей средней
input int    MovingShift    = 6;          // Сдвиг скользящей средней
//--- глобальные переменные
int     IndicatorHandle=0; // хендл индикатора
```

```

bool IsHedging=false; // признак счета
CTrade trade; // для проведения торговых операций
//---
#define EA_MAGIC 18052018
//+-----+
//| Проверка условий на открытие позиции |
//+-----+
void CheckForOpen(void)
{
    MqlRates rt[2];
//--- торгуем только в начале нового бара
    if(CopyRates(_Symbol,_Period,0,2,rt)!=2)
    {
        Print("CopyRates of ",_Symbol," failed, no history");
        return;
    }
//--- тиковый объем
    if(rt[1].tick_volume>1)
        return;
//--- получим значения скользящей средней
    double ma[1];
    if(CopyBuffer(IndicatorHandle,0,1,1,ma)!=1)
    {
        Print("CopyBuffer from iMA failed, no data");
        return;
    }
//--- проверим наличие сигнала
    ENUM_ORDER_TYPE signal=WRONG_VALUE;
//--- свеча открылась выше, а закрылась ниже скользящей средней
    if(rt[0].open>ma[0] && rt[0].close<ma[0])
        signal=ORDER_TYPE_BUY; // сигнал на покупку
    else // свеча открылась ниже, а закрылась выше скользящей средней
    {
        if(rt[0].open<ma[0] && rt[0].close>ma[0])
            signal=ORDER_TYPE_SELL;// сигнал на продажу
    }
//--- дополнительные проверки
    if(signal!=WRONG_VALUE)
    {
        if(TerminalInfoInteger(TERMINAL_TRADE_ALLOWED) && Bars(_Symbol,_Period)>100)
        {
            double price=SymbolInfoDouble(_Symbol,signal==ORDER_TYPE_SELL ? SYMBOL_BID:SYMBOL_ASK);
            trade.PositionOpen(_Symbol,signal,Lots,price,0,0);
        }
    }
//---
}
//+-----+
//| Проверка условий на закрытие позиции |
//+-----+

```

```

//+-----+
void CheckForClose(void)
{
    MqlRates rt[2];
//--- торгуем только в начале нового бара
    if(CopyRates(_Symbol,_Period,0,2,rt)!=2)
    {
        Print("CopyRates of ",_Symbol," failed, no history");
        return;
    }
    if(rt[1].tick_volume>1)
        return;
//--- получим значения скользящей средней
    double ma[1];
    if(CopyBuffer(IndicatorHandle,0,1,1,ma)!=1)
    {
        Print("CopyBuffer from iMA failed, no data");
        return;
    }
//--- позиция уже была выбрана ранее с помощью PositionSelect()
    bool signal=false;
    long type=PositionGetInteger(POSITION_TYPE);
//--- свеча открылась выше, а закрылась ниже скользящей средней - закрываем короткую позицию
    if(type==(long)POSITION_TYPE_SELL && rt[0].open>ma[0] && rt[0].close<ma[0])
        signal=true;
//--- свеча открылась ниже, а закрылась выше скользящей средней - закрываем длинную позицию
    if(type==(long)POSITION_TYPE_BUY && rt[0].open<ma[0] && rt[0].close>ma[0])
        signal=true;
//--- дополнительные проверки
    if(signal)
    {
        if(TerminalInfoInteger(TERMINAL_TRADE_ALLOWED) && Bars(_Symbol,_Period)>100)
            trade.PositionClose(_Symbol,Slippage);
    }
//---
}
//+-----+
//| Выбираем позицию с учетом типа счета: Netting или Hedging |
//+-----+
bool SelectPosition()
{
    bool res=false;
//--- выбор позиции для счета Hedging
    if(IsHedging)
    {
        uint total=PositionsTotal();
        for(uint i=0; i<total; i++)
        {
            string position_symbol=PositionGetSymbol(i);

```

```

        if(_Symbol==position_symbol && EA_MAGIC==PositionGetInteger(POSITION_MAGIC))
        {
            res=true;
            break;
        }
    }
//--- выбор позиции для счета Netting
else
{
    if(!PositionSelect(_Symbol))
        return(false);
    else
        return(PositionGetInteger(POSITION_MAGIC)==EA_MAGIC); //---проверка Magic number
}
//--- результат выполнения
return(res);
}

//-----+
//| Expert initialization function
//-----+
int OnInit(void)
{
//--- установим тип торговли: Netting или Hedging
IsHedging=((ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger(ACCOUNT_MARGIN_MODE)==ACCOUNT_MARGIN_NETTING);
//--- инициализируем объект для правильного контроля позиций
trade.SetExpertMagicNumber(EA_MAGIC);
trade.SetMarginMode();
trade.SetTypeFillingBySymbol(Symbol());
trade.SetDeviationInPoints(Slippage);
//--- создадим индикатор Moving Average
IndicatorHandle=iMA(_Symbol,_Period,MovingPeriod,MovingShift,MODE_SMA,PRICE_CLOSE);
if(IndicatorHandle==INVALID_HANDLE)
{
    printf("Ошибка при создании индикатора iMA");
    return(INIT_FAILED);
}
//--- ok
return(INIT_SUCCEEDED);
}

//-----+
//| Expert tick function
//-----+
void OnTick(void)
{
//--- если позиция уже открыта, то проверим условие на закрытие
if>SelectPosition())
    CheckForClose();
// проверим условие на открытие позиции
}

```

```

CheckForOpen();
//---

}

//+-----+
//| Tester function
//+-----+

double OnTester()
{
//--- значение пользовательского критерия оптимизации (чем больше, тем лучше)
    double ret=0.0;
//--- получим результаты трейдов в массив
    double array[];
    double trades_volume;
    GetTradeResultsToArray(array,trades_volume);
    int trades=ArraySize(array);

//--- если трейдов меньше 10, то тестирование не дало положительных результатов
    if(trades<10)
        return (0);

//--- средний результат на трейд
    double average_pl=0;
    for(int i=0;i<ArraySize(array);i++)
        average_pl+=array[i];
    average_pl/=trades;

//--- выведем сообщение для режима одиночного тестирования
    if(MQLInfoInteger(MQL_TESTER) && !MQLInfoInteger(MQL_OPTIMIZATION))
        PrintFormat("%s: Трейдов=%d, Средняя прибыль=%.2f",__FUNCTION__,trades,average_p
//--- посчитаем коэффициенты линейной регрессии для графика прибыли
    double a,b,std_error;
    double chart[];
    if(!CalculateLinearRegression(array,chart,a,b))
        return (0);

//--- вычислим ошибку отклонения графика от линии регрессии
    if(!CalculateStdError(chart,a,b,std_error))
        return (0);

//--- вычислим отношение трендовой прибыли к среднеквадратичному отклонению
    ret=a*trades/std_error;
//--- вернем значение пользовательского критерия оптимизации
    return(ret);
}

//+-----+
//| Получить массив прибылей/убытков из сделок
//+-----+

bool GetTradeResultsToArray(double &pl_results[],double &volume)
{
//--- запросим полную торговую историю
    if(!HistorySelect(0,TimeCurrent()))
        return (false);
    uint total_deals=HistoryDealsTotal();
    volume=0;
}

```

```

//--- установим начальный размер массива с запасом - по количеству сделок в истории
ArrayResize(pl_results,total_deals);

//--- счетчик сделок, фиксирующих торговый результат - прибыль или убыток
int counter=0;
ulong ticket_history_deal=0;

//--- пройдем по всем сделкам
for(uint i=0;i<total_deals;i++)
{
    //--- выберем сделку
    if((ticket_history_deal=HistoryDealGetTicket(i))>0)
    {
        ENUM_DEAL_ENTRY deal_entry =(ENUM_DEAL_ENTRY)HistoryDealGetInteger(ticket_history_deal,DEAL_ENTRY);
        long deal_type =HistoryDealGetInteger(ticket_history_deal,DEAL_TYPE);
        double deal_profit =HistoryDealGetDouble(ticket_history_deal,DEAL_PROFIT);
        double deal_volume =HistoryDealGetDouble(ticket_history_deal,DEAL_VOLUME);

        //--- нас интересуют только торговые операции
        if((deal_type!=DEAL_TYPE_BUY) && (deal_type!=DEAL_TYPE_SELL))
            continue;

        //--- только сделки с фиксацией прибыли/убытка
        if(deal_entry!=DEAL_ENTRY_IN)
        {
            //--- запишем торговый результат в массив и увеличим счетчик трейдов
            pl_results[counter]=deal_profit;
            volume+=deal_volume;
            counter++;
        }
    }
}

//--- установим окончательный размер массива
ArrayResize(pl_results,counter);
return (true);
}

//+-----+
//| Вычисляет линейную регрессию вида y=a*x+b |
//+-----+
bool CalculateLinearRegression(double &change[],double &chartline[],
                                double &a_coef,double &b_coef)
{
    //--- проверим достаточность данных
    if(ArraySize(change)<3)
        return (false);

    //--- создадим массив графика с накоплением
    int N=ArraySize(change);
    ArrayResize(chartline,N);
    chartline[0]=change[0];
    for(int i=1;i<N;i++)
        chartline[i]=chartline[i-1]+change[i];

    //--- теперь вычислим коэффициенты регрессии
    double x=0,y=0,x2=0,xy=0;
}

```

```
for(int i=0;i<N;i++)
{
    x=x+i;
    y=y+chartline[i];
    xy=xy+i*chartline[i];
    x2=x2+i*i;
}
a_coef=(N*xy-x*y) / (N*x2-x*x);
b_coef=(y-a_coef*x) / N;
//---
return (true);
}

//+-----+
//| Вычисляет среднеквад. ошибку отклонения для заданных а и б |
//+-----+
bool CalculateStdError(double &data[],double a_coef,double b_coef,double &std_err)
{
//--- сумма квадратов ошибки
double error=0;
int N=ArraySize(data);
if(N==0)
    return (false);
for(int i=0;i<N;i++)
    error=MathPow(a_coef*i+b_coef-data[i],2);
std_err=MathSqrt(error/(N-2));
//---
return (true);
}
```

#### Смотри также

[Тестирование торговых стратегий](#), [TesterHideIndicators](#), [Работа с результатами оптимизации](#), [TesterStatistics](#), [OnTesterInit](#), [OnTesterDeinit](#), [OnTesterPass](#), [MQL\\_TESTER](#), [MQL\\_OPTIMIZATION](#), [FileOpen](#), [FileWrite](#), [FileLoad](#), [FileSave](#)

## OnTesterInit

Вызывается в экспертах при наступлении события [TesterInit](#) для выполнения необходимых действий перед началом оптимизации в тестере стратегий. Существуют два варианта функции.

### Версия с возвратом результата

```
int OnTesterInit(void);
```

### Возвращаемое значение

Значение типа [int](#), ноль означает успешную инициализацию эксперта, запущенного на графике перед началом оптимизации.

Приоритетным является использование вызова `OnTesterInit()` с возвратом результата выполнения, так как этот способ позволяет не только выполнить инициализацию программы, но и вернуть код ошибки в случае досрочного прекращения оптимизации. Возврат любого значения, отличного от [INIT\\_SUCCEEDED](#) (0), означает ошибку и оптимизация запущена не будет.

**Версия без возврата результата** оставлена только для совместимости со старыми кодами. Не рекомендуется к использованию

```
void OnTesterInit(void);
```

### Примечание

Событие [TesterInit](#) автоматически генерируется перед началом оптимизации эксперта в тестере стратегий. По данному событию эксперт, имеющий обработчик `OnTesterDeinit()` и/или `OnTesterPass()`, автоматически загружается на отдельном графике терминала с указанными в тестере символом и периодом.

Такой эксперт получает события [TesterInit](#), [TesterDeinit](#) и [TesterPass](#), но не получает событий [Init](#), [Deinit](#) и [NewTick](#). Соответственно, всю необходимую логику по обработке результатов каждого прохода в процессе оптимизации необходимо реализовать в обработчиках `OnTesterInit()`, `OnTesterDeinit()` и `OnTesterPass()`.

Результат каждого одиночного прохода при оптимизации стратегии можно передать через фрейм из обработчика [OnTester\(\)](#) с помощью функции [FrameAdd\(\)](#).

Функция `OnTesterInit()` предназначена для инициализации эксперта перед началом оптимизации для последующей [обработки результатов оптимизации](#). Всегда используется совместно с обработчиком `OnTesterDeinit()`.

На выполнение `OnTesterInit()` отводится ограниченное время, по превышении которого будет произведено принудительное завершение работы эксперта, а сама оптимизация будет отменена. При этом в Журнал тестера будет выведено сообщение:

TesterOnTesterInit works too long. Tester cannot be initialized.

**Пример** взят из [OnTick](#), добавлен обработчик `OnTesterInit()` для установки параметров оптимизации:

```
//+-----+
//|                                     OnTesterInit_Sample.mq5 |
//| Copyright 2018, MetaQuotes Software Corp. |
```

```
//+
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Пример советника с обработчиком OnTesterInit()"
#property description "в котором устанавливаются значения и границы"
#property description "входных параметров при оптимизации"

input double lots=0.1;           // объем в лотах
input double kATR=3;             // длина сигнальной свечи в ATR
input int    ATRperiod=20;        // период индикатора ATR
input int    holdbars=8;          // сколько баров удерживаем позицию
input int    slippage=10;          // допустимое проскальзывание
input bool   revers=false;        // переворачиваем сигнал?
input ulong  EXPERT_MAGIC=0;      // MagicNumber эксперта
//--- для хранения хендла индикатора ATR
int atr_handle;
//--- здесь будем хранить последние значения ATR и тела свечи
double last_atr,last_body;
datetime lastbar_timeopen;
double trade_lot;
//--- запоминаем время начала оптимизации
datetime optimization_start;
//--- для вывода на график длительности после окончании оптимизации
string report;
//+-----+
//| TesterInit function
//+-----+
void OnTesterInit()
{
//--- установим значения входных параметров для оптимизации
ParameterSetRange("lots",false,0.1,0,0,0);
ParameterSetRange("kATR",true,3.0,1.0,0.3,7.0);
ParameterSetRange("ATRperiod",true,10,15,1,30);
ParameterSetRange("holdbars",true,5,3,1,15);
ParameterSetRange("slippage",false,10,0,0,0);
ParameterSetRange("revers",true,false,false,1,true);
ParameterSetRange("EXPERT_MAGIC",false,123456,0,0,0);
Print("Установлены начальные значения и границы параметров оптимизации");
//--- запомним начало оптимизации
optimization_start=TimeLocal();
report=StringFormat("%s: оптимизация запущена в %s",
                     __FUNCTION__, TimeToString(TimeLocal(),TIME_MINUTES|TIME_SECONDS));
//--- выведем сообщения на график и в журнал терминала
Print(report);
Comment(report);
//---
}
```

```

//+-----+
//| TesterDeinit function
//+-----+
void OnTesterDeinit()
{
//--- продолжительность оптимизации
string log_message=StringFormat("%s: оптимизация заняла %d секунды",
                                 __FUNCTION__, TimeLocal()-optimization_start);
PrintFormat(log_message);
report=report+"\r\n"+log_message;
Comment(report);
}

//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- инициализируем глобальные переменные
last_atr=0;
last_body=0;
//--- установим правильный объем
double min_lot=SymbolInfoDouble(_Symbol, SYMBOL_VOLUME_MIN);
trade_lot=lots>min_lot? lots:min_lot;
//--- создадим хендл индикатора ATR
atr_handle=iATR(_Symbol, _Period, ATRperiod);
if(atr_handle==INVALID_HANDLE)
{
    PrintFormat("%s: не удалось создать iATR, код ошибки %d", __FUNCTION__, GetLastError());
    return(INIT_FAILED);
}
//--- успешная инициализация эксперта
return(INIT_SUCCEEDED);
}

//+-----+
//| Expert tick function
//+-----+
void OnTick()
{
//--- торговый сигнал
static int signal=0; // +1 означает сигнал на покупку, -1 означает сигнал на продажу
//--- проверим и закроем старые позиции, открытые более holdbars баров назад
ClosePositionsByBars(holdbars, slippage, EXPERT_MAGIC);
//--- проверим появление нового бара
if(isNewBar())
{
    //--- проверим наличие сигнала
    signal=CheckSignal();
}
//--- если открыта неттинговая позиция, то сигнал пропускаем - ждем, пока она закроется
}

```

```

if(signal!=0 && PositionsTotal ()>0 && (ENUM_ACCOUNT_MARGIN_MODE)AccountInfoInteger
{
    signal=0;
    return; // выходим из обработчика события NewTick и не входим в рынок до появления
}
//--- для хеджингового счета каждая позиция живет и закрывается раздельно
if(signal!=0)
{
    //--- сигнал на покупку
    if(signal>0)
    {
        PrintFormat ("%s: Есть сигнал на покупку! Revers=%s", __FUNCTION__, string(revers));
        if(Buy(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
    }
    //--- сигнал на продажу
    if(signal<0)
    {
        PrintFormat ("%s: Есть сигнал на продажу! Revers=%s", __FUNCTION__, string(revers));
        if(Sell(trade_lot,slippage,EXPERT_MAGIC))
            signal=0;
    }
}
//--- конец функции OnTick
}

//-----+
//| Проверяет наличие торгового сигнала |
//-----+
int CheckSignal()
{
//--- 0 означает отсутствие сигнала
int res=0;
//--- получим значение ATR на предпоследнем завершенном баре (индекс бара равен 2)
double atr_value[1];
if(CopyBuffer(atr_handle,0,2,1,atr_value)!=-1)
{
    last_atr=atr_value[0];
//--- получим данные последнего закрытого бара в массив типа MqlRates
MqlRates bar[1];
if(CopyRates(_Symbol,_Period,1,1,bar)!=-1)
{
    //--- вычислим размер тела бара на последнем закрытом баре
    last_body=bar[0].close-bar[0].open;
//--- если тело последнего бара (с индексом 1) превышает предыдущее значение
    if(MathAbs(last_body)>kATR*last_atr)
        res=last_body>0?1:-1; // для растущей свечи положительное значение
}
else
    PrintFormat ("%s: Не удалось получить последний бар! Ошибка", __FUNCTION__, GetLastError());
}
}

```

```

    }

    else
        PrintFormat("%s: Не удалось получить значение индикатора ATR! Ошибка", __FUNCTION__)
//--- если включен реверсивный режим торговли
    res=revers?-res:res; // если нужно, то развернем сигнал (вместо 1 вернем -1, а вместо
//--- вернем значение торгового сигнала
    return (res);
}

//+-----+
//| Возвращает true при появлении нового бара |
//+-----+

bool isNewBar(const bool print_log=true)
{
    static datetime bartime=0; // храним время открытия текущего бара
//--- получим время открытия нулевого бара
    datetime currbar_time=iTime(_Symbol,_Period,0);
//--- если время открытия изменилось, значит появился новый бар
    if(bartime!=currbar_time)
    {
        bartime=currbar_time;
        lastbar_timeopen=bartime;
//--- нужно ли выводить в лог информацию о времени открытия нового бара
        if(print_log && !(MQLInfoInteger(MQL_OPTIMIZATION) || MQLInfoInteger(MQL_TESTER)))
        {
//--- выведем сообщение о времени открытия нового бара
            PrintFormat("%s: new bar on %s %s opened at %s", __FUNCTION__, _Symbol,
                        StringSubstr(EnumToString(_Period),7),
                        TimeToString(TimeCurrent(),TIME_SECONDS));
//--- получим данные о последнем тике
            MqlTick last_tick;
            if(!SymbolInfoTick(Symbol(),last_tick))
                Print("SymbolInfoTick() failed, error = ",GetLastError());
//--- выведем время последнего тика с точностью до миллисекунд
            PrintFormat("Last tick was at %.%03d",
                        TimeToString(last_tick.time,TIME_SECONDS),last_tick.time_msc%1000);
        }
//--- у нас есть новый бар
        return (true);
    }
//--- нового бара нет
    return (false);
}

//+-----+
//| Покупка по рынку с заданным объемом |
//+-----+

bool Buy(double volume,ulong deviation=10,ulong magicnumber=0)
{
//--- покупаем по рыночной цене
    return (MarketOrder(ORDER_TYPE_BUY,volume,deviation,magicnumber));
}

```

```

    }

//+-----+
//| Продажа по рынку с заданным объемом |
//+-----+

bool Sell(double volume, ulong deviation=10, ulong magicnumber=0)
{
    //--- продаем по рыночной цене
    return (MarketOrder(ORDER_TYPE_SELL, volume, deviation, magicnumber));
}

//+-----+
//| Закрытие позиций по времени удержания в барах |
//+-----+

void ClosePositionsByBars(int holdtimebars, ulong deviation=10, ulong magicnumber=0)
{
    int total=PositionsTotal(); // количество открытых позиций
    //--- перебор всех открытых позиций
    for(int i=total-1; i>=0; i--)
    {
        //--- параметры позиции
        ulong position_ticket=PositionGetTicket(i);
        string position_symbol=PositionGetString(POSITION_SYMBOL);
        ulong magic=PositionGetInteger(POSITION_MAGIC);
        datetime position_open=(datetime)PositionGetInteger(POSITION_TIME);
        int bars=iBarShift(_Symbol, PERIOD_CURRENT, position_open)+1;

        //--- если позиция живет уже долго, а также MagicNumber и символ совпадают
        if(bars>holdtimebars && magic==magicnumber && position_symbol==_Symbol)
        {
            int digits=(int)SymbolInfoInteger(position_symbol, SYMBOL_DIGITS);
            double volume=PositionGetDouble(POSITION_VOLUME);
            ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
            string str_type=StringSubstr(EnumToString(type), 14);
            StringToLower(str_type); // понижаем регистр текста для правильного форматирования
            PrintFormat("Закрываем позицию #%-d %s %s %.2f",
                position_ticket, position_symbol, str_type, volume);
            //--- установка типа ордера и отправки торгового запроса
            if(type==POSITION_TYPE_BUY)
                MarketOrder(ORDER_TYPE_SELL, volume, deviation, magicnumber, position_ticket);
            else
                MarketOrder(ORDER_TYPE_BUY, volume, deviation, magicnumber, position_ticket);
        }
    }
}

//+-----+
//| Подготовка и отправка торгового запроса |
//+-----+

bool MarketOrder(ENUM_ORDER_TYPE type, double volume, ulong slip, ulong magicnumber, ulong
{
    //--- объявление и инициализация структур
}

```

```

MqlTradeRequest request={0};
MqlTradeResult result={0};
double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
if(type==ORDER_TYPE_BUY)
    price=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- параметры запроса
request.action =TRADE_ACTION_DEAL; // тип торговой операции
request.position =pos_ticket; // тикет позиции, если за...
request.symbol =Symbol(); // символ
request.volume =volume; // объем
request.type =type; // тип ордера
request.price =price; // цена совершения сделки
request.deviation=slip; // допустимое отклонение о...
request.magic =magicnumber; // MagicNumber ордера
//--- отправка запроса
if(!OrderSend(request,result))
{
    //--- выведем информацию о неудаче
    PrintFormat("OrderSend %s %s %.2f at %.5f error %d",
               request.symbol,EnumToString(type),volume,request.price,GetLastError);
    return (false);
}
//--- сообщим об успешной операции
PrintFormat("retcode=%u deal=%I64u order=%I64u",result.retcode,result.deal,result.order);
return (true);
}

```

#### Смотри также

[Тестирование торговых стратегий](#), [Работа с результатами оптимизации](#), [OnTesterDeinit](#),  
[OnTesterPass](#), [ParameterGetRange](#), [ParameterSetRange](#)

## OnTesterDeinit

Вызывается в экспертах при наступлении события [TesterDeinit](#) для выполнения необходимых действий по окончании оптимизации эксперта.

```
void OnTesterDeinit(void);
```

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

Событие [TesterDeinit](#) автоматически генерируется по окончании оптимизации эксперта в тестере стратегий.

Эксперт, имеющий обработчик `OnTesterDeinit()` или `OnTesterPass()`, при запуске оптимизации автоматически загружается на отдельном графике терминала с указанными в тестере символом и периодом. Функция предназначена для финальной обработки всех [результатов оптимизации](#).

Необходимо помнить, что фреймы оптимизации, отсылаемые агентами тестирования с помощью функции [FrameAdd\(\)](#), могут приходить пачками и для их доставки требуется время. Поэтому не все фреймы, а соответственно и события [TesterPass](#), могут до окончания оптимизации поступить и быть обработанными в [OnTesterPass\(\)](#). Поэтому для гарантированного получения всех запоздавших фреймов в `OnTesterDeinit()` необходимо поместить блок кода с использованием функции [FrameNext\(\)](#).

### Смотри также

[Тестирование торговых стратегий](#), [Работа с результатами оптимизации](#), [TesterStatistics](#), [OnTesterInit](#), [OnTesterPass](#), [ParameterGetRange](#), [ParameterSetRange](#)

## OnTesterPass

Вызывается в экспертах при наступлении события [TesterPass](#) для обработки нового фрейма данных во время оптимизации эксперта.

```
void OnTesterPass(void);
```

### Возвращаемое значение

Нет возвращаемого значения

### Примечание

Событие [TesterPass](#) автоматически генерируется при поступлении фрейма во время оптимизации эксперта в тестере стратегий.

Эксперт, имеющий обработчик [OnTesterDeinit\(\)](#) или [OnTesterPass\(\)](#), при запуске оптимизации автоматически загружается на отдельном графике терминала с указанными в тестере символом и периодом. Функция предназначена для обработки фреймов, полученных от агентов тестирования во время оптимизации. Отправку фрейма с результатами тестирования необходимо выполнять из обработчика [OnTester\(\)](#) с помощью функции [FrameAdd\(\)](#).

Необходимо помнить, что фреймы оптимизации, отсылаемые агентами тестирования с помощью функции [FrameAdd\(\)](#), могут приходить пачками и для их доставки требуется время. Поэтому не все фреймы, а соответственно и события [TesterPass](#), могут до окончания оптимизации поступить и быть обработанными в [OnTesterPass\(\)](#). Поэтому для гарантированного получения всех запоздавших фреймов в [OnTesterDeinit\(\)](#) необходимо поместить блок кода с использованием функции [FrameNext\(\)](#).

После завершения оптимизации [OnTesterDeinit\(\)](#) можно заново перебрать все полученные фреймы с помощью функций [FrameFirst\(\)](#)/[FrameFilter](#) и [FrameNext\(\)](#).

### Смотри также

[Тестирование торговых стратегий](#), [Работа с результатами оптимизации](#), [OnTesterInit](#), [OnTesterDeinit](#), [FrameFirst](#), [FrameFilter](#), [FrameNext](#), [FrameInputs](#)

## Получение рыночной информации

Функции для получения информации о состоянии рынка.

Функция	Действие
<a href="#">SymbolsTotal</a>	Возвращает количество доступных (выбранных в MarketWatch или всех) символов
<a href="#">SymbolExist</a>	Проверяет наличие символа с указанным именем
<a href="#">SymbolName</a>	Возвращает наименование указанного символа
<a href="#">SymbolSelect</a>	Выбирает символ в окне MarketWatch или убирает символ из окна
<a href="#">SymbolIsSynchronized</a>	Проверяет <a href="#">синхронизированность</a> данных по указанному символу в терминале с данными на торговом сервере
<a href="#">SymbolInfoDouble</a>	Возвращает значение типа double указанного символа для соответствующего свойства
<a href="#">SymbolInfoInteger</a>	Возвращает значение целочисленного типа (long, datetime, int или bool) указанного символа для соответствующего свойства
<a href="#">SymbolInfoString</a>	Возвращает значение типа string указанного символа для соответствующего свойства
<a href="#">SymbolInfoMarginRate</a>	Возвращает коэффициенты взимания маржи в зависимости от типа и направления ордера
<a href="#">SymbolInfoTick</a>	Возвращает текущие цены для указанного символа в переменной типа <a href="#">MqlTick</a>
<a href="#">SymbolInfoSessionQuote</a>	Позволяет получить время начала и время окончания указанной котировочной сессии для указанных символа и дня недели.
<a href="#">SymbolInfoSessionTrade</a>	Позволяет получить время начала и время окончания указанной торговой сессии для указанных символа и дня недели.
<a href="#">MarketBookAdd</a>	Обеспечивает открытие стакана цен по указанному инструменту, а также производит подписку на получение извещений об изменении указанного стакана
<a href="#">MarketBookRelease</a>	Обеспечивает закрытие стакана цен по указанному инструменту, а также отменяет подписку на получение извещений об изменении указанного стакана

MarketBookGet

Возвращает массив структур типа [MqlBookInfo](#), содержащий записи стакана цен указанного символа

## SymbolsTotal

Возвращает количество доступных (выбранных в MarketWatch или всех) символов.

```
int SymbolsTotal(
    bool selected // true - только символы в MarketWatch
);
```

### Параметры

*selected*

[in] Режим запроса. Может принимать значения true или false.

### Возвращаемое значение

Если параметр *selected* равно true, то возвращается количество выбранных в MarketWatch символов. Если значение false, то возвращается общее количество всех символов.

## SymbolExist

Проверяет наличие символа с указанным именем.

```
bool SymbolExist(
    const string name,      // имя символа
    bool& is_custom        // признак пользовательского символа
);
```

### Параметры

*name*

[in] Имя символа.

*is\_custom*

[out] Признак пользовательского символа, который выставляется при успешном выполнении.  
Если значение равно true, то найденный символ является [пользовательским](#).

### Возвращаемое значение

Возвращает false, если символ не найден ни среди стандартных, ни среди [пользовательских символов](#).

### Смотри также

[SymbolsTotal](#), [SymbolSelect](#), [Пользовательские символы](#)

## SymbolName

Возвращает наименование указанного символа.

```
string SymbolName(
    int   pos,           // номер в списке
    bool  selected       // true - только символы в MarketWatch
);
```

### Параметры

*pos*

[in] Номер символа по порядку.

*selected*

[in] Режим запроса. Если значение `true`, то символ берется из списка выбранных в MarketWatch. Если значение `false`, то символ берется из общего списка.

### Возвращаемое значение

Значение типа `string` с именем символа.

## SymbolSelect

Выбирает символ в окне MarketWatch (Обзор рынка) или убирает символ из этого окна.

```
bool SymbolSelect(
    string name,           // имя символа
    bool select            // включить или выключить
);
```

### Параметры

*name*

[in] Имя символа.

*select*

[in] Переключатель. Если значение `false`, то символ должен быть убран из окна MarketWatch, в противном случае символ должен быть выбран в окно MarketWatch. Символ не может быть убран, если есть открытые графики с этим символом или есть открытые позиции по этому символу.

### Возвращаемое значение

В случае неудачи функция возвращает `false`.

## SymbolsSynchronized

Проверяет факт синхронизированности данных по указанному символу в терминале с данными на торговом сервере

```
bool SymbolIsSynchronized(  
    string name,           // имя символа  
);
```

### Параметры

*name*

[in] Имя символа.

### Возвращаемое значение

Если данные [синхронизированы](#), возвращает true, иначе возвращает false.

### Смотри также

[SymbolInfoInteger](#), [Организация доступа к данным](#)

## SymbolInfoDouble

Возвращает соответствующее свойство указанного символа. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
double SymbolInfoDouble(
    string           name,          // символ
    ENUM_SYMBOL_INFO_DOUBLE prop_id // идентификатор свойства
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool SymbolInfoDouble(
    string           name,          // символ
    ENUM_SYMBOL_INFO_DOUBLE prop_id, // идентификатор свойства
    double&          double_var   // сюда примем значение свойства
);
```

### Параметры

*name*

[in] Имя символа.

*prop\_id*

[in] Идентификатор свойства символа. Значение может быть одним из значений перечисления [ENUM\\_SYMBOL\\_INFO\\_DOUBLE](#).

*double\_var*

[out] Переменная типа double, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа double. В случае неудачного выполнения информацию об [ошибке](#) можно получить с помощью функции [GetLastError\(\)](#):

- 5040 - неверный строковый параметр для указания имени символа,
- 4301 - неизвестный символ (финансовый инструмент),
- 4302 - символ не выбран в "Обзоре рынка" (нет в списке доступных),
- 4303 - неверный идентификатор свойства символа.

### Примечание

Если функция используется для получения информации о последнем тике, то лучше использовать [SymbolInfoTick\(\)](#). Вполне возможно, что по данному символу с момента подключения терминала к торговому счету не было еще ни одной котировки. В таком случае запрашиваемое значение будет неопределенным.

В большинстве случаев достаточно использовать функцию [SymbolInfoTick\(\)](#), которая позволяет получить за один вызов значения Ask, Bid, Last, Volume и время прихода последнего тика.

### Пример:

```
void OnTick()
{
//--- получим спред из свойств символа
bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
string comm=StringFormat("Спред %s = %I64d пунктов\r\n",
                           spreadfloat?"плавающий":"фиксированный",
                           SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- вычислим теперь спред сами
double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
double spread=ask-bid;
int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBOL_POINT));
comm=comm+"Вычисленный спред = "+(string)spread_points+" пунктов";
Comment(comm);
}
```

## SymbolInfoInteger

Возвращает соответствующее свойство указанного символа. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
long SymbolInfoInteger(
    string             name,           // символ
    ENUM_SYMBOL_INFO_INTEGER prop_id   // идентификатор свойства
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool SymbolInfoInteger(
    string             name,           // символ
    ENUM_SYMBOL_INFO_INTEGER prop_id, // идентификатор свойства
    long&              long_var     // сюда примем значение свойства
);
```

### Параметры

*name*

[in] Имя символа.

*prop\_id*

[in] Идентификатор свойства символа. Значение может быть одним из значений перечисления [ENUM\\_SYMBOL\\_INFO\\_INTEGER](#).

*long\_var*

[out] Переменная типа long, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа long. В случае неудачного выполнения информацию об [ошибке](#) можно получить с помощью функции [GetLastError\(\)](#):

- 5040 - неверный строковый параметр для указания имени символа,
- 4301 - неизвестный символ (финансовый инструмент),
- 4302 - символ не выбран в "Обзоре рынка" (нет в списке доступных),
- 4303 - неверный идентификатор свойства символа.

### Примечание

Если функция используется для получения информации о последнем тике, то лучше использовать [SymbolInfoTick\(\)](#). Вполне возможно, что по данному символу с момента подключения терминала к торговому счету не было еще ни одной котировки. В таком случае запрашиваемое значение будет неопределенным.

В большинстве случаев достаточно использовать функцию [SymbolInfoTick\(\)](#), которая позволяет получить за один вызов значения Ask, Bid, Last, Volume и время прихода последнего тика.

### Пример:

```
void OnTick()
{
//--- получим спред из свойств символа
bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
string comm=StringFormat("Спред %s = %I64d пунктов\r\n",
                           spreadfloat?"плавающий":"фиксированный",
                           SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- вычислим теперь спред сами
double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
double spread=ask-bid;
int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBOL_POINT));
comm=comm+"Вычисленный спред = "+(string)spread_points+" пунктов";
Comment(comm);
}
```

## SymboInfoString

Возвращает соответствующее свойство указанного символа. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
string SymbolInfoString(
    string name, // символ
    ENUM_SYMBOL_INFO_STRING prop_id // идентификатор свойства
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool SymbolInfoString(
    string name, // символ
    ENUM_SYMBOL_INFO_STRING prop_id, // идентификатор свойства
    string& string_var // сюда примем значение свойства
);
```

### Параметры

*name*

[in] Имя символа.

*prop\_id*

[in] Идентификатор свойства символа. Значение может быть одним из значений перечисления [ENUM\\_SYMBOL\\_INFO\\_STRING](#).

*string\_var*

[out] Переменная типа string, принимающая значение запрашиваемого свойства. В случае неудачного выполнения информацию об ошибке можно получить с помощью функции [GetLastError\(\)](#):

- 5040 - неверный строковый параметр для указания имени символа,
- 4301 - неизвестный символ (финансовый инструмент),
- 4302 - символ не выбран в "Обзоре рынка" (нет в списке доступных),
- 4303 - неверный идентификатор свойства символа.

### Примечание

Если функция используется для получения информации о последнем тике, то лучше использовать [SymbolInfoTick\(\)](#). Вполне возможно, что по данному символу с момента подключения терминала к торговому счету не было еще ни одной котировки. В таком случае запрашиваемое значение будет неопределенным.

В большинстве случаев достаточно использовать функцию [SymbolInfoTick\(\)](#), которая позволяет получить за один вызов значения Ask, Bid, Last, Volume и время прихода последнего тика.

### Возвращаемое значение

Значение типа string.

## SymbolInfoMarginRate

Возвращает коэффициенты взимания маржи в зависимости от типа и направления ордера.

```
bool SymbolInfoMarginRate(
    string          name,           // символ
    ENUM_ORDER_TYPE order_type,    // тип ордера
    double&         initial_margin_rate, // коэффициент взимания начальной маржи
    double&         maintenance_margin_rate // коэффициент взимания поддерживающей маржи
);
```

### Параметры

*name*

[in] Имя символа.

*order\_type*

[in] Тип ордера.

*initial\_margin\_rate*

[in] Переменная типа [double](#) для получения коэффициента взимания начальной маржи. Начальная маржа - это размер гарантийной суммы под совершение сделки объемом в 1 лот соответствующего направления. Умножая коэффициент на начальную маржу, мы можем получить размер средств, который будет зарезервирован на счете при размещении ордера указанного типа.

*maintenance\_margin\_rate*

[out] Переменная типа [double](#) для получения коэффициента взимания поддерживающей маржи. Поддерживающая маржа - это размер минимальной суммы для поддержания открытой позиции объемом в 1 лот соответствующего направления. Умножая коэффициент на поддерживающую маржу, мы можем получить размер средств, который будет зарезервирован на счете после срабатывания ордера указанного типа.

### Возвращаемое значение

Возвращает `true` в случае удачного выполнения запроса свойств, иначе `false`.

## SymbolInfoTick

Возвращает текущие цены для указанного символа в переменной типа [MqlTick](#).

```
bool SymbolInfoTick(  
    string     symbol,      // символ  
    MqlTick&  tick        // ссылка на структуру  
);
```

### Параметры

*symbol*

[in] Имя символа.

*tick*

[out] Ссылка на структуру типа [MqlTick](#), в которую будут помещены текущие цены и время последнего обновления цен.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## SymbolInfoSessionQuote

Позволяет получить время начала и время окончания указанной котировочной сессии для указанных символа и дня недели.

```
bool SymbolInfoSessionQuote(
    string          name,           // имя символа
    ENUM_DAY_OF_WEEK day_of_week,   // день недели
    uint            session_index, // номер сессии
    datetime&       from,          // время начала сессии
    datetime&       to             // время окончания сессии
);
```

### Параметры

*name*

[in] Имя символа.

*ENUM\_DAY\_OF\_WEEK*

[in] День недели, значение из перечисления [ENUM\\_DAY\\_OF\\_WEEK](#).

*uint*

[in] Порядковый номер сессии, для которой нужно получить время начала и время окончания.  
Индексация сессий начинается с 0.

*from*

[out] Время начала сессии в секундах от 00 часов 00 минут, в полученном значении дату следует игнорировать.

*to*

[out] Время окончания сессии в секундах от 00 часов 00 минут, в полученном значении дату следует игнорировать.

### Возвращаемое значение

Если данные для указанных сессии, символа и дня недели получены, то возвращает true, иначе возвращает false.

### Смотри также

[Информация об инструменте](#), [TimeToStruct](#), [Структура даты](#)

## SymbolInfoSessionTrade

Позволяет получить время начала и время окончания указанной торговой сессии для указанных символа и дня недели.

```
bool SymbolInfoSessionTrade(
    string          name,           // имя символа
    ENUM_DAY_OF_WEEK day_of_week,   // день недели
    uint            session_index, // номер сессии
    datetime&       from,          // время начала сессии
    datetime&       to             // время окончания сессии
);
```

### Параметры

*name*

[in] Имя символа.

*ENUM\_DAY\_OF\_WEEK*

[in] День недели, значение из перечисления [ENUM\\_DAY\\_OF\\_WEEK](#).

*uint*

[in] Порядковый номер сессии, для которой нужно получить время начала и время окончания.  
Индексация сессий начинается с 0.

*from*

[out] Время начала сессии в секундах от 00 часов 00 минут, в полученном значении дату следует игнорировать.

*to*

[out] Время окончания сессии в секундах от 00 часов 00 минут, в полученном значении дату следует игнорировать.

### Возвращаемое значение

Если данные для указанных сессии, символа и дня недели получены, то возвращает true, иначе возвращает false.

### Смотри также

[Информация об инструменте](#), [TimeToStruct](#), [Структура даты](#)

## MarketBookAdd

Обеспечивает открытие стакана цен по указанному инструменту, а также производит подписку на получение извещений об изменении указанного стакана.

```
bool MarketBookAdd(  
    string symbol // символ  
) ;
```

### Параметры

*symbol*

[in] Имя символа, чей стакан цен предполагается использовать в данном эксперте или скрипте.

### Возвращаемое значение

Значение true в случае успешного открытия, иначе false.

### Примечание

Обычно, эта функция должна вызываться из функции [OnInit\(\)](#) или в конструкторе класса. Для обработки приходящих извещений в программе эксперта должна присутствовать функция [void OnBookEvent\(string& symbol\)](#).

### Смотри также

[Структура стакана цен](#), [Структуры и классы](#)

## MarketBookRelease

Обеспечивает закрытие стакана цен по указанному инструменту, а также отменяет подписку на получение извещений об изменении указанного стакана.

```
bool MarketBookRelease(  
    string symbol // имя символа  
) ;
```

### Параметры

*symbol*  
[in] Имя символа.

### Возвращаемое значение

Значение true в случае успешного закрытия, иначе false.

### Примечание

Обычно эта функция должна вызываться из функции [OnDeinit\(\)](#) в том случае, если в функции [OnInit\(\)](#) была вызвана соответствующая функция [MarketBookAdd\(\)](#). Либо должна вызываться из деструктора класса, если в конструкторе этого класса вызывается соответствующая функция [MarketBookAdd\(\)](#).

### Смотри также

[Структура стакана цен](#), [Структуры и классы](#)

## MarketBookGet

Возвращает массив структур [MqlBookInfo](#), содержащий записи стакана цен указанного символа.

```
bool MarketBookGet(
    string      symbol,      // символ
    MqlBookInfo& book[]     // ссылка на массив
);
```

### Параметры

*symbol*

[in] Имя символа.

*book []*

[out] Ссылка на массив записей стакана цен. Массив может быть заранее распределен для достаточного количества записей. Если [динамический массив](#) не был заранее распределен в оперативной памяти, то клиентский терминал сам распределит этот массив.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Стакан цен должен быть предварительно открыт функцией [MarketBookAdd\(\)](#).

### Пример:

```
MqlBookInfo priceArray[];
bool getBook=MarketBookGet(NULL,priceArray);
if(getBook)
{
    int size=ArraySize(priceArray);
    Print("MarketBookInfo по ",Symbol());
    for(int i=0;i<size;i++)
    {
        Print(i+":",priceArray[i].price
              +" Volume= "+priceArray[i].volume,
              " type = ",priceArray[i].type);
    }
}
else
{
    Print("Не удалось получить содержимое стакана по символу ",Symbol());
}
```

### Смотри также

[Структура стакана цен](#), [Структуры и классы](#)

## Функции Экономического Календаря

В этом разделе описываются функции для работы с [Экономическим календарем](#), который доступен прямо в платформе MetaTrader. Экономический календарь является готовой энциклопедией с описанием макроэкономических индикаторов, даты их выхода и степени важности. Актуальные значения макроэкономических показателей поступают в платформу MetaTrader моментально прямо в момент публикации и отображаются на графике в виде меток - это позволяет визуально отслеживать нужные показатели в разрезе стран, валют и важности.

Все функции для работы с Экономическим календарем используют время торгового сервера ([TimeTradeServer](#)). Это означает, что время в структуре [MqlCalendarValue](#) и входящие параметры времени в функциях [CalendarValueHistoryByEvent](#)/[CalendarValueHistory](#) задаются в таймзоне торгового сервера, а не в локальном времени пользователя.

Функции [Экономического календаря](#) позволяют проводить автоматический анализ поступающих событий по собственным критериям важности и в разрезе нужных стран/валют.

Функция	Действие
<a href="#">CalendarCountryById</a>	Получает описание страны по её идентификатору
<a href="#">CalendarEventById</a>	Получает описание события по его идентификатору
<a href="#">CalendarValueById</a>	Получает описание значения события по его идентификатору
<a href="#">CalendarCountries</a>	Получает массив описаний стран, доступных в Календаре
<a href="#">CalendarEventByCountry</a>	Получает массив описаний всех событий, доступных в Календаре, по указанному коду страны
<a href="#">CalendarEventByCurrency</a>	Получает массив описаний всех событий, доступных в Календаре, по указанной валюте
<a href="#">CalendarValueHistoryByEvent</a>	Получает массив значений по всем событиям на заданном диапазоне времени по идентификатору события
<a href="#">CalendarValueHistory</a>	Получает массив значений по всем событиям на заданном диапазоне времени с фильтром по стране и/или валюте
<a href="#">CalendarValueLastByEvent</a>	Получает массив значений события по его ID с момента состояния базы Календаря с заданным change_id
<a href="#">CalendarValueLast</a>	Получает массив значений по всем событиям с фильтрацией по стране и/или валюте с момента состояния базы Календаря с заданным change_id

## CalendarCountryById

Получает описание страны по её идентификатору.

```
bool CalendarCountryById(
    const long          country_id,      // идентификатор страны
    MqlCalendarCountry& country        // переменная для получения описания страны
);
```

### Параметры

*country\_id*

[in] Идентификатор страны по стандарту [ISO 3166-1](#).

*country*

[out] Переменная типа [MqlCalendarCountry](#) для получения описания страны.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4001 - ERR\_INTERNAL\_ERROR (общая ошибка исполняющей системы),
- 5402 - ERR\_CALENDAR\_NO\_DATA (страна не найдена),
- 5401 - ERR\_CALENDAR\_TIMEOUT (превышен лимит запроса по времени).

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- получим список стран из Экономического календаря
MqlCalendarCountry countries[];
int count=CalendarCountries(countries);
//--- проверим результат
if(count==0)
    PrintFormat("CalendarCountries() returned 0! Error %d",GetLastError());
//--- если у нас две и более стран
if(count>=2)
{
    MqlCalendarCountry country;
//--- теперь получим описание страны по её идентификатору
if(CalendarCountryById(countries[1].id, country))
{
//--- подготовим описание страны
string descr="id = "+IntegerToString(country.id)+"\n";
descr+="name = " + country.name+"\n";
descr+="code = " + country.code+"\n";
descr+="currency = " + country.currency+"\n";
descr+="currency_symbol = " + country.currency_symbol+"\n";
}
```

```
descr+="url_name = " + country.url_name);
//---выведем описание страны
Print(descr);
}
else
    Print("CalendarCountryById() failed. Error ",GetLastError());
}
//---
}
/*
Результат:
id = 999
name = European Union
code = EU
currency = EUR
currency_symbol = €
url_name = european-union
*/
```

#### Смотри также

[CalendarCountries](#), [CalendarEventByCountry](#)

## CalendarEventById

Получает описание события по его идентификатору.

```
bool CalendarEventById(
    ulong           event_id,      // идентификатор события
    MqlCalendarEvent& event        // переменная для получения описания события
);
```

### Параметры

*event\_id*

[in] Идентификатор события.

*event*

[out] Переменная типа [MqlCalendarEvent](#) для получения описания события.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4001 - ERR\_INTERNAL\_ERROR (общая ошибка исполняющей системы),
- 5402 - ERR\_CALENDAR\_NO\_DATA (страна не найдена),
- 5401 - ERR\_CALENDAR\_TIMEOUT (превышен лимит запроса по времени).

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- код страны для Германии по стандарту ISO 3166-1 Alpha-2
    string germany_code="DE";
//--- получим события для Германии
    MqlCalendarEvent events[];
    int events_count=CalendarEventByCountry(germany_code,events);
//--- выведем события для Германии в Журнал
    if(events_count>0)
    {
        PrintFormat("События для Германии: %d",events_count);
        ArrayPrint(events);
    }
    else
    {
        PrintFormat("Не удалось получить события для кода страны %s, ошибка %d",
                    germany_code,GetLastError());
//--- досрочное завершение скрипта
        return;
    }
//--- получим описание последнего события из массива events[]
}
```

```

MqlCalendarEvent event;
ulong event_id=events[events_count-1].id;
if(CalendarEventById(event_id,event))
{
    MqlCalendarCountry country;
    CalendarCountryById(event.country_id,country);
    PrintFormat("Получено описания события с event_id=%d",event_id);
    PrintFormat("Страна: %s (код страны = %d)",country.name,event.country_id);
    PrintFormat("Имя события: %s",event.name);
    PrintFormat("Код события: %s",event.event_code);
    PrintFormat("Важность события: %s",EnumToString((ENUM_CALENDAR_EVENT_IMPORTANCE));
    PrintFormat("Тип события: %s",EnumToString((ENUM_CALENDAR_EVENT_TYPE)event.type);
    PrintFormat("Сектор события: %s",EnumToString((ENUM_CALENDAR_EVENT_SECTOR)event.
    PrintFormat("Периодичность события: %s",EnumToString((ENUM_CALENDAR_EVENT_FREQUEN
    PrintFormat("Режим выхода события: %s",EnumToString((ENUM_CALENDAR_EVENT_TIMEMODE));
    PrintFormat("Единица измерения значения: %s",EnumToString((ENUM_CALENDAR_EVENT_UNIT));
    PrintFormat("Количество знаков после запятой: %d",event.digits);
    PrintFormat("Множитель значения: %s",EnumToString((ENUM_CALENDAR_EVENT_MULTIPLIER));
    PrintFormat("URL источника: %s",event.source_url);
}
else
{
    PrintFormat("Не удалось получить описание события для event_id=%s, ошибка %d",
               event_id,GetLastError());
}
/*
Результат:
События для Германии: 50
[id] [type] [sector] [frequency] [time_mode] [country_id] [unit] [importance]
[ 0] 276010001   1      6      2      0      276      1
[ 1] 276010002   1      6      2      0      276      1
[ 2] 276010003   1      4      2      0      276      1
[ 3] 276010004   1      4      2      0      276      1
...
[47] 276500001   1      8      2      0      276      0
[48] 276500002   1      8      2      0      276      0
[49] 276500003   1      8      2      0      276      0
Получено описания события с event_id=276500003
Страна: Germany (код страны = 276)
Имя события: Markit Composite PMI
Код события: markit-composite-pmi
Важность события: CALENDAR_IMPORTANCE_MODERATE
Тип события: CALENDAR_TYPE_INDICATOR
Сектор события: CALENDAR_SECTOR_BUSINESS
Периодичность события: CALENDAR_FREQUENCY_MONTH
Режим выхода события: CALENDAR_TIMEMODE_DATETIME
Единица измерения значения: CALENDAR_UNIT_NONE
Количество знаков после запятой: 1
Множитель значения: CALENDAR_MULTIPLIER_NONE
URL источника: https://www.markiteconomics.com

```

\*/

**Смотри также**[CalendarEventByCountry](#), [CalendarEventByCurrency](#), [CalendarValueById](#)

## CalendarValueById

Получает описание значения события по его идентификатору.

```
bool CalendarValueById(
    ulong           value_id,      // идентификатор значения события
    MqlCalendarValue& value        // переменная для получения значения события
);
```

### Параметры

`value_id`

[in] Идентификатор значения события.

`value`

[out] Переменная типа [MqlCalendarValue](#) для получения значения события.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4001 - ERR\_INTERNAL\_ERROR (общая ошибка исполняющей системы),
- 5402 - ERR\_CALENDAR\_NO\_DATA (страна не найдена),
- 5401 - ERR\_CALENDAR\_TIMEOUT (превышен лимит запроса по времени).

### Примечание

Все функции для работы с Экономическим календарем используют время торгового сервера ([TimeTradeServer](#)). Это означает, что время в структуре [MqlCalendarValue](#) и входящие параметры времени в функциях [CalendarValueHistoryByEvent](#)/[CalendarValueHistory](#) задаются в таймзоне торгового сервера, а не в локальном времени пользователя.

### Пример:

```
//+-----+
//| Script program start function          |
//+-----+
void OnStart()
{
//--- код страны для Японии по стандарту ISO 3166-1 Alpha-2
    string japan_code="JP";
//--- зададим границы диапазона, из которого берем события
    datetime date_from=D'01.01.2018'; // берем все события с 2018 года
    datetime date_to=0;                // 0 означает все известные события, даже те, что
//--- получим массив значений событий для Японии
    MqlCalendarValue values[];
    int values_count=CalendarValueHistory(values,date_from,date_to,japan_code);
//--- пройдем по найденным значениям событий
    if(values_count>0)
    {
        PrintFormat("Количество значений для событий Японии: %d",values_count);
    }
}
```

```

//--- удалим все "пустые" значения (actual_value===-9223372036854775808)
for(int i=values_count-1;i>=0;i--)
{
    if(values[i].actual_value===-9223372036854775808)
        ArrayRemove(values,i,1);
}
PrintFormat("Количество значений после удаления пустых: %d",ArraySize(values));
}
else
{
    PrintFormat("Не удалось получить события для кода страны %s, ошибка %d",
                japan_code,GetLastError());
    //--- досрочное завершение скрипта
    return;
}
//--- оставим не более 10 значений в массиве values[]
if(ArraySize(values)>10)
{
    PrintFormat("Уменьшим список значений до 10 и покажем их");
    ArrayRemove(values,0,ArraySize(values)-10);
}
ArrayPrint(values);

//--- теперь покажем как на основе известного value_id получить описание значения события
for(int i=0;i<ArraySize(values);i++)
{
    MqlCalendarValue value;
    CalendarValueById(values[i].id,value);
    PrintFormat("%d: value_id=%d value=%d impact=%s",
               i,values[i].id,value.actual_value,EnumToString(ENUM_CALENDAR_EVENT_IMPACT,
               value.event_impact));
}
//---
}
/*
Результат:
Количество значений для событий Японии: 1734
Количество значений после удаления пустых: 1017
Уменьшим список значений до 10 и покажем их
[ id] [event_id] [time] [period] [revision] [actual_value]
[0] 56500 392030004 2019.03.28 23:30:00 2019.03.01 00:00:00 0 900
[1] 56501 392030005 2019.03.28 23:30:00 2019.03.01 00:00:00 0 700
[2] 56502 392030006 2019.03.28 23:30:00 2019.03.01 00:00:00 0 1100
[3] 56544 392030007 2019.03.28 23:30:00 2019.02.01 00:00:00 0 2300
[4] 56556 392050002 2019.03.28 23:30:00 2019.02.01 00:00:00 0 1630
[5] 55887 392020003 2019.03.28 23:50:00 2019.02.01 00:00:00 0 400
[6] 55888 392020004 2019.03.28 23:50:00 2019.02.01 00:00:00 0 -1800
[7] 55889 392020002 2019.03.28 23:50:00 2019.02.01 00:00:00 0 200
[8] 55948 392020006 2019.03.28 23:50:00 2019.02.01 00:00:00 1 1400
[9] 55949 392020007 2019.03.28 23:50:00 2019.02.01 00:00:00 1 -1000

```

```
Выведем краткую информацию по значениям событий на основе value_id
0: value_id=56500 value=900000 impact=CALENDAR_IMPACT_POSITIVE
1: value_id=56501 value=700000 impact=CALENDAR_IMPACT_NA
2: value_id=56502 value=1100000 impact=CALENDAR_IMPACT_POSITIVE
3: value_id=56544 value=2300000 impact=CALENDAR_IMPACT_NEGATIVE
4: value_id=56556 value=1630000 impact=CALENDAR_IMPACT_POSITIVE
5: value_id=55887 value=400000 impact=CALENDAR_IMPACT_NEGATIVE
6: value_id=55888 value=-1800000 impact=CALENDAR_IMPACT_POSITIVE
7: value_id=55889 value=200000 impact=CALENDAR_IMPACT_NEGATIVE
8: value_id=55948 value=1400000 impact=CALENDAR_IMPACT_POSITIVE
9: value_id=55949 value=-1000000 impact=CALENDAR_IMPACT_NEGATIVE
*/
```

#### Смотри также

[CalendarValueHistoryByEvent](#), [CalendarValueHistory](#), [CalendarValueLastByEvent](#), [CalendarValueLast](#)

## CalendarCountries

Получает массив описаний стран, доступных в Календаре.

```
int CalendarCountries(
    MqlCalendarCountry& countries[]           // массив для получения списка описаний стран
);
```

### Параметры

*countries[]*

[out] Массив типа [MqlCalendarCountry](#) для получения описаний всех стран Календаря.

### Возвращаемое значение

Количество полученных описаний. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4001 - ERR\_INTERNAL\_ERROR (общая ошибка исполняющей системы),
- 5401 - ERR\_CALENDAR\_TIMEOUT (превышен лимит запроса по времени),
- 5400 - ERR\_CALENDAR\_MORE\_DATA (размер массива недостаточен для получения описаний всех стран, поэтому получили только то, что вместилось).

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- получим список стран из Экономического календаря
    MqlCalendarCountry countries[];
    int count=CalendarCountries(countries);
//--- выведем массив в Журнал
    if(count>0)
        ArrayPrint(countries);
    else
        PrintFormat("CalendarCountries() returned 0! Error %d",GetLastError());
/*
```

#### Результат:

[id]	[name]	[code]	[currency]	[currency_symbol]	[url_name]	[re]
[ 0 ]	0 "Worldwide"	"WW"	"ALL"	""	"worldwide"	
[ 1 ]	999 "European Union"	"EU"	"EUR"	"€"	"european-union"	
[ 2 ]	840 "United States"	"US"	"USD"	"\$"	"united-states"	
[ 3 ]	124 "Canada"	"CA"	"CAD"	"\$"	"canada"	
[ 4 ]	36 "Australia"	"AU"	"AUD"	"\$"	"australia"	
[ 5 ]	554 "New Zealand"	"NZ"	"NZD"	"\$"	"new-zealand"	
[ 6 ]	392 "Japan"	"JP"	"JPY"	"¥"	"japan"	
[ 7 ]	156 "China"	"CN"	"CNY"	"¥"	"china"	
[ 8 ]	826 "United Kingdom"	"GB"	"GBP"	"£"	"united-kingdom"	
[ 9 ]	756 "Switzerland"	"CH"	"CHF"	"₣"	"switzerland"	

```
[10] 276 "Germany"      "DE"   "EUR"   "€"        "germany"
[11] 250 "France"       "FR"   "EUR"   "€"        "france"
[12] 380 "Italy"        "IT"   "EUR"   "€"        "italy"
[13] 724 "Spain"         "ES"   "EUR"   "€"        "spain"
[14] 76  "Brazil"        "BR"   "BRL"   "R$"      "brazil"
[15] 410 "South Korea"  "KR"   "KRW"   "₩"       "south-korea"
*/
}
```

Смотри также

[CalendarEventByCountry](#), [CalendarCountryById](#)

## CalendarEventByCountry

Получает массив описаний всех событий, доступных в Календаре, по указанному коду страны.

```
int CalendarEventByCountry(
    string          country_code,      // кодовое имя страны по ISO 3166-1 alpha-2
    MqlCalendarEvent& events[]        // переменная для получения массива описаний
);
```

### Параметры

*country\_code*

[in] Кодовое имя страны согласно ISO 3166-1 alpha-2

*events[]*

[out] Массив типа [MqlCalendarEvent](#) для получения описаний всех событий для указанной страны.

### Возвращаемое значение

Количество полученных описаний. Чтобы получить информацию об ошибке, необходимо вызывать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4001 - ERR\_INTERNAL\_ERROR (общая ошибка исполняющей системы),
- 4004 - ERR\_NOT\_ENOUGH\_MEMORY (не достаточно памяти для выполнения запроса),
- 5401 - ERR\_CALENDAR\_TIMEOUT (превышен лимит запроса по времени),
- ошибки неудачного выполнения [ArrayResize\(\)](#)

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- код страны для Европейского союза по стандарту ISO 3166-1 Alpha-2
    string EU_code="EU";
//--- получим события для Евросоюза
    MqlCalendarEvent events[];
    int events_count=CalendarEventByCountry(EU_code,events);
//--- выведем события для Евросоюза в Журнал
    if(events_count>0)
    {
        PrintFormat("События для Европейского союза: %d",events_count);
        ArrayPrint(events);
    }
//---
}
/*
Результат:
События для Европейского союза: 56
[id] [type] [country_id] [unit] [importance] [multiplier] [digits] [ever
```

[ 0]	999010001	0	999	0	2	0	0	"ECB
[ 1]	999010002	0	999	0	2	0	0	"ECB
[ 2]	999010003	0	999	0	3	0	0	"ECB
[ 3]	999010004	0	999	0	3	0	0	"ECB
[ 4]	999010005	0	999	0	2	0	0	"ECB
[ 5]	999010006	1	999	1	3	0	2	"ECB
[ 6]	999010007	1	999	1	3	0	2	"ECB
[ 7]	999010008	0	999	0	2	0	0	"ECB
[ 8]	999010009	1	999	2	2	3	3	"ECB
[ 9]	999010010	0	999	0	2	0	0	"ECB
[10]	999010011	0	999	0	2	0	0	"ECB
...								
*/								

Смотри также

[CalendarCountries](#), [CalendarCountryByld](#)

## CalendarEventByCurrency

Получает массив описаний всех событий, доступных в Календаре, по указанной валюте.

```
int CalendarEventByCurrency(
    const string      currency,      // кодовое наименование валюты страны
    MqlCalendarEvent& events[]       // переменная для получения массива описаний
);
```

### Параметры

*currency*

[in] Кодовое наименование валюты страны.

*events[]*

[out] Массив типа [MqlCalendarEvent](#) для получения описаний всех событий для указанной валюты.

### Возвращаемое значение

Количество полученных описаний. Чтобы получить информацию об ошибке, необходимо вызывать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4001 - ERR\_INTERNAL\_ERROR (общая ошибка исполняющей системы),
- 4004 - ERR\_NOT\_ENOUGH\_MEMORY (не достаточно памяти для выполнения запроса),
- 5401 - ERR\_CALENDAR\_TIMEOUT (превышен лимит запроса по времени),
- ошибки неудачного выполнения [ArrayResize\(\)](#)

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- объявим массив для получения событий Экономического календаря
MqlCalendarEvent events[];
//--- получим события для валюты Европейского союза
int count = CalendarEventByCurrency("EUR",events);
Print("count = ", count);
//--- для примера нам достаточно 10 событий
if(count>10)
    ArrayResize(events,10);
//--- выведем события в Журнал
ArrayPrint(events);
}
/*
Результат:
[id] [type] [country_id] [unit] [importance]
[0] 999010001 0 999 0 2 "https://www.ecb.europa.eu/ho
[1] 999010002 0 999 0 2 "https://www.ecb.europa.eu/ho
```

[2]	999010003	0	999	0	3	"https://www.ecb.europa.eu/ho
[3]	999010004	0	999	0	3	"https://www.ecb.europa.eu/ho
[4]	999010005	0	999	0	2	"https://www.ecb.europa.eu/ho
[5]	999010006	1	999	1	3	"https://www.ecb.europa.eu/ho
[6]	999010007	1	999	1	3	"https://www.ecb.europa.eu/ho
[7]	999010008	0	999	0	2	"https://www.ecb.europa.eu/ho
[8]	999010009	1	999	2	2	"https://www.ecb.europa.eu/ho
[9]	999010010	0	999	0	2	"https://www.ecb.europa.eu/ho
*/						

Смотри также

[CalendarEventById](#), [CalendarEventByCountry](#)

## CalendarValueHistoryByEvent

Получает массив значений по всем событиям на заданном диапазоне времени по идентификатору события.

```
bool CalendarValueHistoryByEvent(
    ulong          event_id,           // идентификатор события
    MqlCalendarValue& values[],        // массив для получения описаний значений
    datetime       datetime_from,       // левая граница диапазона времени
    datetime       datetime_to=0         // правая граница диапазона времени
);
```

### Параметры

*event\_id*

[in] Идентификатор события.

*values[]*

[out] Массив типа [MqlCalendarValue](#) для получения значений событий.

*datetime\_from*

[in] Начальная дата диапазона времени, из которого выбираются события по заданному идентификатору, при этом *datetime\_from* < *datetime\_to*.

*datetime\_to=0*

[in] Конечная дата диапазона времени, из которого выбираются события по заданному идентификатору. Если параметр *datetime\_to* не задан (или равен 0), то будут возвращены все значения событий от заданной даты *datetime\_from*, имеющиеся в базе Календаря, в том числе и значения событий, которые запланированы на будущее.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4001 - ERR\_INTERNAL\_ERROR (общая ошибка исполняющей системы),
- 4004 - ERR\_NOT\_ENOUGH\_MEMORY (не достаточно памяти для выполнения запроса),
- 5401 - ERR\_CALENDAR\_TIMEOUT (превышен лимит запроса по времени),
- 5400 - ERR\_CALENDAR\_MORE\_DATA (размер массива недостаточен для получения описаний всех значений, поэтому получили только то, что вместилось),
- ошибки неудачного выполнения [ArrayResize\(\)](#)

Если для значения события отсутствует какое либо из полей ниже

```
struct MqlCalendarValue
{
    ...
    long      actual_value;           // актуальное значение события
    long      prev_value;            // предыдущее значение события
    long      revised_prev_value;    // пересмотренное предыдущее значение
    long      forecast_value;        // прогнозное значение события
    ...
};
```

то значение отсутствующего поля вернется как INT64\_MIN (-9223372036854775808). Смотри в примере ниже значения поля *revised\_prev\_value*.

### Примечание

Все функции для работы с Экономическим календарем используют время торгового сервера ([TimeTradeServer](#)). Это означает, что время в структуре [MqlCalendarValue](#) и входящие параметры времени в функциях [CalendarValueHistoryByEvent/CalendarValueHistory](#) задаются в таймзоне торгового сервера, а не в локальном времени пользователя.

### Пример:

```
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- код страны для Европейского союза по стандарту ISO 3166-1 Alpha-2
    string EU_code="EU";
//--- получим события для Евросоюза
    MqlCalendarEvent events[];
    int events_count=CalendarEventByCountry(EU_code,events);
//--- выведем события для Евросоюза в Журнал
    if(events_count>0)
    {
        PrintFormat("События для Европейского союза: %d",events_count);
        //--- сократим список событий, для изучения достаточно оставить 10
        ArrayResize(events,10);
        ArrayPrint(events);
    }
//--- видим, что событие "ECB Interest Rate Decision" имеет event_id=999010007
    ulong event_id=events[6].id;          // id этого события может поменяться в Календаре
    string event_name=events[6].name;     // имя события из Календаря
    PrintFormat("Получим значения для события event_name=%s event_id=%d",event_name,event_id);
//--- получим все значения события "ECB Interest Rate Decision"
    MqlCalendarValue values[];
//--- зададим границы диапазона, из которого берем события
    datetime date_from=0;                // берем все события с начала доступной истории
    datetime date_to=D'01.01.2016'; // берем события не старше 2016 года
    if(CalendarValueHistoryByEvent(event_id,values,date_from,date_to))
    {
        PrintFormat("Получены значения события %s: %d",
                    event_name,ArraySize(values));
        //--- сократим список значений, для изучения достаточно оставить 10
        ArrayResize(values,10);
        ArrayPrint(values);
    }
else
{
    PrintFormat("Ошибка! Не удалось получить значения для события event_id=%d",event_id);
}
```

```

        PrintFormat("Код ошибки: %d", GetLastError());
    }
}
//---
/*
Результат:
События для Европейского союза: 56
[ id] [type] [sector] [frequency] [time_mode] [country_id] [unit] [importan
[0] 999010001      0      5          0          0         999      0
[1] 999010002      0      5          0          0         999      0
[2] 999010003      0      5          0          0         999      0
[3] 999010004      0      5          0          0         999      0
[4] 999010005      0      5          0          0         999      0
[5] 999010006      1      5          0          0         999      1
[6] 999010007      1      5          0          0         999      1
[7] 999010008      0      5          0          0         999      0
[8] 999010009      1      5          0          0         999      2
[9] 999010010      0      5          0          0         999      0
Получим значения для события event_name=ECB Interest Rate Decision event_id=999010007
Получены значения события ECB Interest Rate Decision: 102
[ id] [event_id]           [time]           [period] [revision] [actual_valu
[0] 2776 999010007 2007.03.08 11:45:00 1970.01.01 00:00:00      0     37500
[1] 2777 999010007 2007.05.10 11:45:00 1970.01.01 00:00:00      0     37500
[2] 2778 999010007 2007.06.06 11:45:00 1970.01.01 00:00:00      0     40000
[3] 2779 999010007 2007.07.05 11:45:00 1970.01.01 00:00:00      0     40000
[4] 2780 999010007 2007.08.02 11:45:00 1970.01.01 00:00:00      0     40000
[5] 2781 999010007 2007.09.06 11:45:00 1970.01.01 00:00:00      0     40000
[6] 2782 999010007 2007.10.04 11:45:00 1970.01.01 00:00:00      0     40000
[7] 2783 999010007 2007.11.08 12:45:00 1970.01.01 00:00:00      0     40000
[8] 2784 999010007 2007.12.06 12:45:00 1970.01.01 00:00:00      0     40000
[9] 2785 999010007 2008.01.10 12:45:00 1970.01.01 00:00:00      0     40000
*/

```

### Смотри также

[CalendarCountries](#), [CalendarEventByCountry](#), [CalendarValueHistory](#), [CalendarEventById](#),  
[CalendarValueById](#)

## CalendarValueHistory

Получает массив значений по всем событиям на заданном диапазоне времени с фильтром по стране и/или валюте.

```
bool CalendarValueHistory(
    MqlCalendarValue& values[],           // массив для получения описаний значений
    datetime      datetime_from,          // левая граница диапазона времени
    datetime      datetime_to=0,           // правая граница диапазона времени
    const string   country_code=NULL,       // кодовое имя страны по ISO 3166-1 alpha-2
    const string   currency=NULL          // кодовое наименование валюты страны
);
```

### Параметры

*values[]*

[out] Массив типа [MqlCalendarValue](#) для получения значений событий.

*datetime\_from*

[in] Начальная дата диапазона времени, из которого выбираются события по заданному идентификатору, при этом *datetime\_from* < *datetime\_to*.

*datetime\_to=0*

[in] Конечная дата диапазона времени, из которого выбираются события по заданному идентификатору. Если параметр *datetime\_to* не задан (или равен 0), то будут возвращены все значения событий от заданной даты *datetime\_from*, имеющиеся в базе Календаря, в том числе и значения событий, которые запланированы на будущее.

*country\_code=NULL*

[in] Кодовое имя страны согласно ISO 3166-1 alpha-2

*currency=NULL*

[in] Кодовое наименование валюты страны.

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4001 - `ERR_INTERNAL_ERROR` (общая ошибка исполняющей системы),
- 4004 - `ERR_NOT_ENOUGH_MEMORY` (не достаточно памяти для выполнения запроса),
- 5401 - `ERR_CALENDAR_TIMEOUT` (превышен лимит запроса по времени),
- 5400 - `ERR_CALENDAR_MORE_DATA` (размер массива недостаточен для получения описаний всех значений, поэтому получили только то, что вместилось),
- ошибки неудачного выполнения [ArrayResize\(\)](#)

### Примечание

Все функции для работы с Экономическим календарем используют время торгового сервера ([TimeTradeServer](#)). Это означает, что время в структуре [MqlCalendarValue](#) и входящие параметры времени в функциях [CalendarValueHistoryByEvent](#)/[CalendarValueHistory](#) задаются в таймзоне торгового сервера, а не в локальном времени пользователя.

Если в функцию был передан массив `events[]` фиксированной длины и в результате запроса не хватило места для сохранения всего результата, будет взведена ошибка `ERR_CALENDAR_MORE_DATA` (5400).

Если не задан параметр `datetime_to` (=0), то будут возвращены все значения событий от заданной даты `datetime_from`, имеющиеся в базе Календаря, в том числе и значения, которые запланированы на будущее.

Для фильтров `country_code` и `currency` значения `NULL` и `""` равносильны - означают отсутствие фильтра.

Для `country_code` следует использовать поле `code` структуры [MqlCalendarCountry](#), например "US", "RU" или "EU".

Для `currency` следует использовать поле `currency` структуры [MqlCalendarCountry](#), например "USD", "RUB" или "EUR".

Фильтры применяются конъюнцией, т.е. через [логическое 'И'](#) выбираются значения только тех событий, для которых удовлетворяются одновременно оба условия - страна и валюта.

Если для значения события отсутствует какое либо из полей ниже

```
struct MqlCalendarValue
{
    ...
    long actual_value;           // актуальное значение события
    long prev_value;            // предыдущее значение события
    long revised_prev_value;    // пересмотренное предыдущее зна
    long forecast_value;        // прогнозное значение события
    ...
};
```

то значение отсутствующего поля вернется как `INT64_MIN` (-9223372036854775808). Смотри в примере ниже значения поля `revised_prev_value`.

**Пример:**

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    //--- код страны для Европейского союза по стандарту ISO 3166-1 Alpha-2
    string EU_code="EU";
    //--- получим все значения событий по Европейскому союзу
    MqlCalendarValue values[];
    //--- зададим границы диапазона, из которого берем события
    datetime date_from=D'01.01.2018'; // берем все события с 2018 года
    datetime date_to=0;                // 0 означает все известные события, даже те, что
    //--- запросим историю события по Европейскому союзу с 2018 года
    if(CalendarValueHistory(values,date_from,date_to,EU_code))
    {
        PrintFormat("Получены значения событий по country_code=%s: %d",
                    EU_code, values[0].revised_prev_value);
    }
}
```

```

        EU_code,ArraySize(values));
//--- уменьшим размер массива для вывода в Журнал
ArrayResize(values,10);
//--- выведем значения событий в Журнал
ArrayPrint(values);
}
else
{
    PrintFormat("Ошибка! Не удалось получить события по стране country_code=%s",EU_code);
    PrintFormat("Код ошибки: %d",GetLastError());
}
//---
}
/*
Результат:
Получены значения событий по country_code=EU: 1384
[ id] [event_id] [time] [period] [revision] [actual_value]
[0] 54215 999500001 2018.01.02 09:00:00 2017.12.01 00:00:00 3 60600
[1] 54221 999500002 2018.01.04 09:00:00 2017.12.01 00:00:00 3 56600
[2] 54222 999500003 2018.01.04 09:00:00 2017.12.01 00:00:00 3 58100
[3] 45123 999030005 2018.01.05 10:00:00 2017.11.01 00:00:00 0 60000
[4] 45124 999030006 2018.01.05 10:00:00 2017.11.01 00:00:00 0 28000
[5] 45125 999030012 2018.01.05 10:00:00 2017.12.01 00:00:00 1 90000
[6] 45126 999030013 2018.01.05 10:00:00 2017.12.01 00:00:00 1 14000
[7] 54953 999520001 2018.01.05 20:30:00 2018.01.02 00:00:00 0 127900
[8] 22230 999040003 2018.01.08 10:00:00 2017.12.01 00:00:00 0 91000
[9] 22231 999040004 2018.01.08 10:00:00 2017.12.01 00:00:00 0 18400
*/

```

#### Смотри также

[CalendarCountries](#), [CalendarEventByCountry](#), [CalendarValueHistoryByEvent](#), [CalendarEventById](#),  
[CalendarValueById](#)

## CalendarValueLastByEvent

Получает массив значений события по его ID с момента состояния базы Календаря с заданным *change\_id*.

```
int CalendarValueLastByEvent(
    ulong          event_id,           // идентификатор события
    ulong&         change_id,          // идентификатор значения события
    MqlCalendarValue& values[]        // массив для получения описаний значений
);
```

### Параметры

*event\_id*

[in] Идентификатор события.

*change\_id*

[in][out] Идентификатор изменения.

*values[]*

[out] Массив типа [MqlCalendarValue](#) для получения значений события.

### Возвращаемое значение

Количество полученных значений события. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4001 - ERR\_INTERNAL\_ERROR (общая ошибка исполняющей системы),
- 4004 - ERR\_NOT\_ENOUGH\_MEMORY (не достаточно памяти для выполнения запроса),
- 5401 - ERR\_CALENDAR\_TIMEOUT (превышен лимит запроса по времени),
- 5400 - ERR\_CALENDAR\_MORE\_DATA (размер массива недостаточен для получения описаний всех значений, поэтому получили только то, что вместилось),
- ошибки неудачного выполнения [ArrayResize\(\)](#)

### Примечание

Все функции для работы с Экономическим календарем используют время торгового сервера ([TimeTradeServer](#)). Это означает, что время в структуре [MqlCalendarValue](#) и входящие параметры времени в функциях [CalendarValueHistoryByEvent](#)/[CalendarValueHistory](#) задаются в таймзоне торгового сервера, а не в локальном времени пользователя.

Если в функцию был передан массив *events[]* фиксированной длины и в результате запроса не хватило места для сохранения всего результата, будет взведена ошибка ERR\_CALENDAR\_MORE\_DATA (5400).

Если в функцию передан *change\_id* = 0, то функция всегда возвращает ноль, но при этом в *change\_id* возвращается текущее состояние базы Календаря.

Функция возвращает массив значений для заданной новости и новое *change\_id*, которые можно использовать для последующих вызовов данной функции, чтобы получить новые значения новости. Таким образом, вызывая эту функцию с последним известным *change\_id*, можно получать обновления значений для заданной новости.

Если для значения события отсутствует какое либо из полей ниже

```

struct MqlCalendarValue
{
    ...
    long           actual_value;          // актуальное значение события
    long           prev_value;           // предыдущее значение события
    long           revised_prev_value;   // пересмотренное предыдущее зна
    long           forecast_value;       // прогнозное значение события
    ...
};

```

то значение отсутствующего поля вернется как INT64\_MIN (-9223372036854775808).

#### Пример эксперта, который слушает выход отчета по Nonfarm payrolls:

```

#property description "Пример использования функции CalendarValueLastByEvent"
#property description "для отлавливания выхода отчета по событию Nonfarm Payrolls."
#property description "Для этого необходимо получить текущий идентификатор изменения"
#property description "базы Календаря. И затем по этому идентификатору получать"
#property description "только новые события для через опрос в таймере"
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- create timer
    EventSetTimer(60);
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- destroy timer
    EventKillTimer();
}
//+-----+
//| Expert tick function
//+-----+
void OnTick()
{
//---
}
//+-----+
//| Timer function
//+-----+
void OnTimer()
{

```

```

//--- идентификатор изменения базы Календаря
static ulong calendar_change_id=0;
//--- признак первого запуска
static bool first=true;
//--- идентификатор события
static ulong event_id=0;
//--- имя события
static string event_name=NULL;
//--- массив значений событий
MqlCalendarValue values[];
//--- проведем инициализацию - получим текущий calendar_change_id
if(first)
{
    MqlCalendarEvent events[];
    //--- код страны для США по стандарту ISO 3166-1 Alpha-2
    string USA_code="US";
    //--- получим события для США
    int events_count=CalendarEventByCountry(USA_code,events);
    //--- позиция нужного нам события в массиве events
    int event_pos=-1;
    //--- выведем события для США в Журнал
    if(events_count>0)
    {
        PrintFormat("%s: События для США: %d",__FUNCTION__,events_count);
        for(int i=0;i<events_count;i++)
        {
            string event_name_low=events[i].name;
            //--- приведем к нижнему регистру имя события
            if(!StringToLower(event_name_low))
            {
                PrintFormat("StringToLower() вернула ошибку %d",GetLastError());
                //--- досрочно выходим из функции
                return;
            }
            //--- поищем событие "Nonfarm Payrolls"
            if(StringFind(event_name_low,"nonfarm payrolls")!=1)
            {
                //--- событие найдено, запомним его id
                event_id=events[i].id;
                //--- запишем для события "Nonfarm Payrolls" его имя
                event_name=events[i].name;
                //--- запомним позицию событий в массиве events[]
                event_pos=i;
                //--- на самом деле в Календаре есть несколько событий, которые содержат
                PrintFormat("Событие \"Nonfarm Payrolls\" найдено: event_id=%d event_name=%s",event_id,event_name);
                //--- посмотрите все события, закомментировав ниже оператор break, чтобы увидеть все
                break;
            }
        }
    }
}

```

```

//--- сократим список, удалим события после события "Nonfarm Payrolls"
ArrayRemove(events,event_pos+1);
//--- для удобства изучения оставим 9 событий перед "Nonfarm Payrolls"
ArrayRemove(events,0,event_pos-9);
ArrayPrint(events);
}

else
{
PrintFormat("%s: CalendarEventByCountry(%s) вернуло 0 событий, код ошибки=%d",
USA_code,__FUNCTION__,GetLastError());
//--- неудачное завершение работы, попробуем заново при следующем вызове таймера
return;
}

//--- получим для указанного события идентификатор изменения базы Календаря
if(CalendarValueLastByEvent(event_id,calendar_change_id,values)>0)
{
//--- этот блок кода не может выполниться при первом запуске, но мы его все равно напишем
PrintFormat("%s: Получен текущий идентификатор базы Календаря: change_id=%d",
__FUNCTION__,calendar_change_id);
//--- выставим флаг и выйдем до следующего события таймера
first=false;
return;
}
else
{
//--- данные не получены (для первого запуска это нормально), проверим наличие ошибки
int error_code=GetLastError();
if(error_code==0)
{
PrintFormat("%s: Получен текущий идентификатор базы Календаря: change_id=%d",
__FUNCTION__,calendar_change_id);
//--- выставим флаг и выйдем до следующего события таймера
first=false;
//--- теперь у нас есть значение calendar_change_id
return;
}
else
{
//--- а вот это действительно ошибка
PrintFormat("%s: Не удалось получить значения для события event_id=%d",__FUNCTION__);
PrintFormat("Код ошибки: %d",error_code);
//--- неудачное завершение работы, попробуем заново при следующем вызове таймера
return;
}
}
}

//--- у нас есть последнее известное значение идентификатора изменения Календаря (change_id)

```

```

ulong old_change_id=calendar_change_id;
//--- проверим - не появилось ли новое значение события "Nonfarm Payrolls"
if(CalendarValueLastByEvent(event_id,calendar_change_id,values)>0)
{
    PrintFormat("%s: Получены новые события для \"%s\": %d",
               __FUNCTION__,event_name,ArraySize(values));
    //--- выведем в Журнал информацию из массива values
    ArrayPrint(values);
    //--- выведем в Журнал значения предыдущего и нового идентификатора Календаря
    PrintFormat("%s: Предыдущий change_id=%d, новый change_id=%d",
               __FUNCTION__,old_change_id,calendar_change_id);
/*
    пропишите здесь свой код, который будет обрабатывать публикацию данных по "Nonfarm Payrolls"
*/
}
/*
*/
//---
}
/*
Результат:
OnTimer: События для США: 202
Событие "Nonfarm Payrolls" найдено: event_id=840030016 event_name=Nonfarm Payrolls
[id] [type] [sector] [frequency] [time_mode] [country_id] [unit] [importance]
[0] 840030007 1 4 2 0 840 1
[1] 840030008 1 4 2 0 840 1
[2] 840030009 1 4 2 0 840 0
[3] 840030010 1 4 2 0 840 0
[4] 840030011 1 4 2 0 840 1
[5] 840030012 1 4 2 0 840 1
[6] 840030013 1 4 2 0 840 1
[7] 840030014 1 4 2 0 840 1
[8] 840030015 1 3 2 0 840 1
[9] 840030016 1 3 2 0 840 4
OnTimer: Получен текущий идентификатор базы Календаря: change_id=33986560
*/

```

## Смотри также

[CalendarValueLast](#), [CalendarValueHistory](#), [CalendarValueHistoryByEvent](#), [CalendarValueById](#)

## CalendarValueLast

Получает массив значений по всем событиям с фильтрацией по стране и/или валюте с момента состояния базы Календаря с заданным `change_id`.

```
int CalendarValueLast(
    ulong&           change_id,          // идентификатор изменения
    MqlCalendarValue& values[],           // массив для получения описаний значений
    const string       country_code=NULL,   // кодовое имя страны по ISO 3166-1 alpha-2
    const string       currency=NULL        // кодовое наименование валюты страны
);
```

### Параметры

`change_id`

[in][out] Идентификатор изменения.

`values[]`

[out] Массив типа [MqlCalendarValue](#) для получения значений события.

`country_code=NULL`

[in] Кодовое имя страны согласно ISO 3166-1 alpha-2

`currency=NULL`

[in] Кодовое наименование валюты страны.

### Возвращаемое значение

Количество полученных значений события. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Возможные ошибки:

- 4001 - `ERR_INTERNAL_ERROR` (общая ошибка исполняющей системы),
- 4004 - `ERR_NOT_ENOUGH_MEMORY` (не достаточно памяти для выполнения запроса),
- 5401 - `ERR_CALENDAR_TIMEOUT` (превышен лимит запроса по времени),
- 5400 - `ERR_CALENDAR_MORE_DATA` (размер массива недостаточен для получения описаний всех значений, поэтому получили только то, что вместилось),
- ошибки неудачного выполнения [ArrayResize\(\)](#)

### Примечание

Все функции для работы с Экономическим календарем используют время торгового сервера ([TimeTradeServer](#)). Это означает, что время в структуре [MqlCalendarValue](#) и входящие параметры времени в функциях [CalendarValueHistoryByEvent](#)/[CalendarValueHistory](#) задаются в таймзоне торгового сервера, а не в локальном времени пользователя.

Если в функцию был передан массив `events[]` фиксированной длины и в результате запроса не хватило места для сохранения всего результата, будет взведена ошибка `ERR_CALENDAR_MORE_DATA` (5400).

Если в функцию передан `change_id = 0`, то функция всегда возвращает ноль, но при этом в `change_id` возвращается текущее состояние базы Календаря.

Для фильтров `country_code` и `currency` значения `NULL` и `""` равносильны - означают отсутствие фильтра.

Для `country_code` следует использовать поле `code` структуры [MqlCalendarCountry](#), например "US", "RU" или "EU".

Для `currency` следует использовать поле `currency` структуры [MqlCalendarCountry](#), например "USD", "RUB" или "EUR".

Фильтры применяются конъюнцией, т.е. через [логическое 'И'](#) выбираются значения только тех событий, для которых удовлетворяются одновременно оба условия - страна и валюта

Функция возвращает массив значений для заданной новости и новое `change_id`, которые можно использовать для последующих вызовов данной функции, чтобы получить новые значения новости. Таким образом, вызывая эту функцию с последним известным `change_id`, можно получать обновления значений для заданной новости.

Если для значения события отсутствует какое либо из полей ниже

```
struct MqlCalendarValue
{
    ...
    long           actual_value;          // актуальное значение события
    long           prev_value;           // предыдущее значение события
    long           revised_prev_value;   // пересмотренное предыдущее зна
    long           forecast_value;       // прогнозное значение события
    ...
};
```

то значение отсутствующего поля вернется как `INT64_MIN (-9223372036854775808)`.

**Пример эксперта, который слушает появление событий Экономического календаря:**

```
#property description "Пример использования функции CalendarValueLast"
#property description " для создания слушателя событий Экономического календаря."
#property description " Для этого необходимо получить текущий идентификатор изменения"
#property description " базы Календаря. И затем по этому идентификатору получать"
#property description " только новые события для через опрос в таймере"
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- create timer
    EventSetTimer(60);
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- destroy timer
    EventKillTimer();
```

```

}

//+-----+
//| Expert tick function
//+-----+
void OnTick()
{
//---

}

//+-----+
//| Timer function
//+-----+
void OnTimer()
{
//--- идентификатор изменения базы Календаря
    static ulong calendar_change_id=0;
//--- признак первого запуска
    static bool first=true;
//--- массив значений событий
    MqlCalendarValue values[];
//--- проведем инициализацию - получим текущий calendar_change_id
    if(first)
    {
//--- получим идентификатор изменения базы Календаря
        if(CalendarValueLast(calendar_change_id,values)>0)
        {
//--- этот блок кода не может выполниться при первом запуске, но мы его все равно выставим
            PrintFormat("%s: Получен текущий идентификатор базы Календаря: change_id=%d",
                        __FUNCTION__,calendar_change_id);
//--- выставим флаг и выйдем до следующего события таймера
            first=false;
            return;
        }
    else
    {
//--- данные не получены (для первого запуска это нормально), проверим наличие ошибки
        int error_code=GetLastError();
        if(error_code==0)
        {
            PrintFormat("%s: Получен текущий идентификатор базы Календаря: change_id=%d",
                        __FUNCTION__,calendar_change_id);
//--- выставим флаг и выйдем до следующего события таймера
            first=false;
//--- теперь у нас есть значение calendar_change_id
            return;
        }
    else
    {
//--- а вот это действительно ошибка
    }
}
}

```

```

PrintFormat("%s: Не удалось получить события в CalendarValueLast. Код ошибки: %d", __FUNCTION__, error_code);
//--- неудачное завершение работы, попробуем провести инициализацию заново
return;
}

}

//--- у нас есть последнее известное значение идентификатора изменения Календаря (change_id)
ulong old_change_id=calendar_change_id;
//--- проверим - не появились ли новые события Календаря
if(CalendarValueLast(calendar_change_id,values)>0)
{
    PrintFormat("%s: Получены новые события Календаря: %d", __FUNCTION__, ArraySize(values));
    //--- выведем в Журнал информацию из массива values
    ArrayPrint(values);
    //--- выведем в Журнал значения предыдущего и нового идентификатора Календаря
    PrintFormat("%s: Предыдущий change_id=%d, новый change_id=%d", __FUNCTION__, old_change_id, calendar_change_id);
    //--- выведем в Журнал новые события
    ArrayPrint(values);
/*
    пропишите здесь свой код, который будет обрабатывать появление событий
*/
}
//---
}
/*
Пример работы слушателя:
OnTimer: Получен текущий идентификатор базы Календаря: change_id=33281792
OnTimer: Получены новые события для Календаря: 1
[id] [event_id] [time] [period] [revision] [actual_value]
[0] 91040 76020013 2019.03.20 15:30:00 1970.01.01 00:00:00 0 -507
OnTimer: Предыдущий change_id=33281792, новый change_id=33282048
[id] [event_id] [time] [period] [revision] [actual_value]
[0] 91040 76020013 2019.03.20 15:30:00 1970.01.01 00:00:00 0 -507
OnTimer: Получены новые события для Календаря: 1
[id] [event_id] [time] [period] [revision] [actual_value]
[0] 91041 76020013 2019.03.27 15:30:00 1970.01.01 00:00:00 0 -9223372036
OnTimer: Предыдущий change_id=33282048, новый change_id=33282560
[id] [event_id] [time] [period] [revision] [actual_value]
[0] 91041 76020013 2019.03.27 15:30:00 1970.01.01 00:00:00 0 -9223372036
*/

```

**Смотри также**

[CalendarValueLast](#), [CalendarValueHistory](#), [CalendarValueHistoryByEvent](#), [CalendarValueById](#)

## Доступ к таймсерием и данным индикаторов

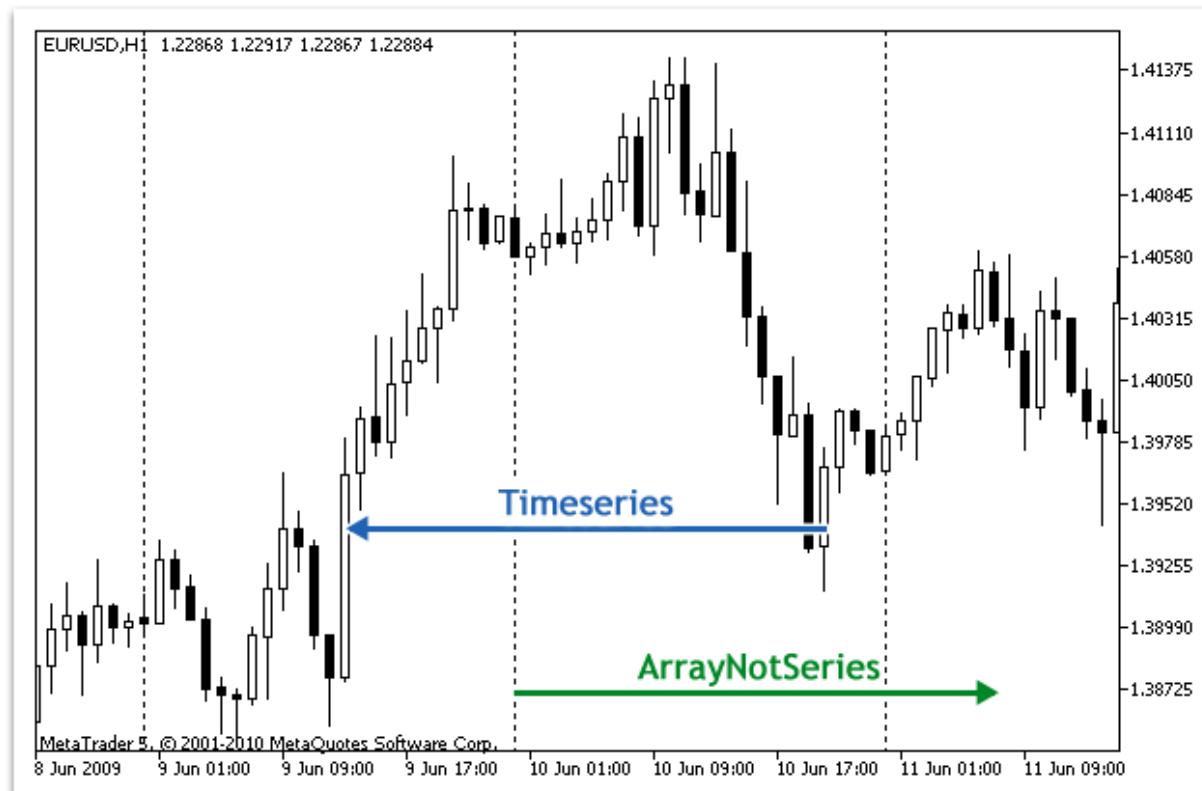
Функции для работы с таймсериеми и индикаторами. Таймсериал отличается от обычного массива тем, что индексация элементов таймсерием производится от конца массива к началу (от самых свежих данных к самым старым). Для копирования значений таймсериал и индикаторов рекомендуется использовать только [динамические массивы](#), так как функции копирования самостоятельно распределяют необходимый размер массивов-приемников значений.

Из этого правила есть **важное исключение**: если копирование таймсериал и значений индикаторов необходимо делать часто, например, при каждом вызове [OnTick\(\)](#) в экспертах или при каждом вызове [OnCalculate\(\)](#) в индикаторах, то в этом случае лучше использовать [статически распределенные массивы](#), так как **операции распределения памяти под динамические массивы требуют дополнительного времени** и это скажется при тестировании и оптимизации экспертов.

При использовании функций доступа к таймсериал и значениям индикаторов необходимо учитывать направление индексации, это подробно описано в разделе [Направление индексации в массивах и таймсерием](#).

Доступ к данным индикаторов и таймсериал осуществляется независимо от факта готовности запрашиваемых данных (так называемый [асинхронный доступ](#)). Это критически важно для расчета пользовательских индикаторов, поэтому при отсутствии запрашиваемых данных функции типа [Copy...\(\)](#) сразу же возвращают ошибку. Однако при доступе из экспертов и скриптов производится несколько попыток получения данных с небольшой паузой, призванной обеспечить время, необходимое для загрузки недостающих таймсериал либо для расчета значений индикаторов.

В разделе [Организация доступа к данным](#) дается подробное описание тонкостей получения, хранения и запроса ценовых данных в клиентском терминале MetaTrader 5.



Исторически сложилось так, что доступ к данным в ценовом массиве производился с конца данных. Физически новые данные всегда дописываются в конец массива, но индекс этого массива всегда равен нулю. Индекс 0 в массиве-таймсерию означает данные текущего бара, то есть бара, который соответствует незавершенному промежутку времени на данном таймфрейме.

Таймфрейм - период времени, в течение которого формируется один ценовой бар; всего предопределен 21 [стандартный таймфрейм](#).

Функция	Действие
<a href="#">SeriesInfoInteger</a>	Возвращает информацию о состоянии исторических данных
<a href="#">Bars</a>	Возвращает количество баров в истории по соответствующим символу и периоду
<a href="#">BarsCalculated</a>	Возвращает количество рассчитанных данных в индикаторном буфере или -1 в случае ошибки (данные еще не рассчитаны)
<a href="#">IndicatorCreate</a>	Возвращает хэндл указанного технического индикатора, созданного на основе массива параметров типа <a href="#">MqlParam</a>
<a href="#">IndicatorParameters</a>	Возвращает по указанному хэндлу количество входных параметров индикатора, а также сами значения и тип параметров
<a href="#">IndicatorRelease</a>	Удаляет хэндл индикатора и освобождает расчетную часть индикатора, если ею больше никто не пользуется
<a href="#">CopyBuffer</a>	Получает в массив данные указанного буфера от указанного индикатора
<a href="#">CopyRates</a>	Получает в массив исторические данные структуры <a href="#">Rates</a> для указанных символа и периода
<a href="#">CopyTime</a>	Получает в массив исторические данные по времени открытия баров по соответствующим символу и периоду
<a href="#">CopyOpen</a>	Получает в массив исторические данные по цене открытия баров по соответствующим символу и периоду
<a href="#">CopyHigh</a>	Получает в массив исторические данные по максимальной цене баров по соответствующим символу и периоду
<a href="#">CopyLow</a>	Получает в массив исторические данные по минимальной цене баров по соответствующим символу и периоду

<a href="#"><u>CopyClose</u></a>	Получает в массив исторические данные по цене закрытия баров по соответствующим символу и периоду
<a href="#"><u>CopyTickVolume</u></a>	Получает в массив исторические данные по тиковым объемам для соответствующих символа и периода
<a href="#"><u>CopyRealVolume</u></a>	Получает в массив исторические данные по торговым объемам для соответствующих символа и периода
<a href="#"><u>CopySpread</u></a>	Получает в массив исторические данные по спредам для соответствующих символа и периода
<a href="#"><u>CopyTicks</u></a>	Получает в массив тики в формате MqlTick
<a href="#"><u>CopyTicksRange</u></a>	Получает в массив тики в указанном диапазоне дат
<a href="#"><u>iBars</u></a>	Возвращает количество баров в истории по соответствующему символу и периоду
<a href="#"><u>iBarShift</u></a>	Возвращает индекс бара, в который попадает указанное время
<a href="#"><u>iClose</u></a>	Возвращает значение цены закрытия бара (указанного параметром shift) соответствующего графика
<a href="#"><u>iHigh</u></a>	Возвращает значение максимальной цены бара (указанного параметром shift) соответствующего графика
<a href="#"><u>iHighest</u></a>	Возвращает индекс наибольшего найденного значения (смещение относительно текущего бара) соответствующего графика
<a href="#"><u>iLow</u></a>	Возвращает значение минимальной цены бара (указанного параметром shift) соответствующего графика
<a href="#"><u>iLowest</u></a>	Возвращает индекс наименьшего найденного значения (смещение относительно текущего бара) соответствующего графика
<a href="#"><u>iOpen</u></a>	Возвращает значение цены открытия бара (указанного параметром shift) соответствующего графика
<a href="#"><u>iTime</u></a>	Возвращает значение времени открытия бара (указанного параметром shift) соответствующего графика

<u>iTickVolume</u>	Возвращает значение тикового объема бара (указанного параметром shift) соответствующего графика
<u>iRealVolume</u>	Возвращает значение реального объема бара (указанного параметром shift) соответствующего графика
<u>iVolume</u>	Возвращает значение тикового объема бара (указанного параметром shift) соответствующего графика
<u>iSpread</u>	Возвращает значение спреда бара (указанного параметром shift) соответствующего графика

Несмотря на то, что функцией [ArraySetAsSeries\(\)](#) можно задавать [массивам](#) способ доступа к элементам как для таймсериий, нужно помнить, что физически элементы массива всегда хранятся в одном и том же порядке, меняется только направление индексации. Для демонстрации этого факта можно выполнить пример:

```

datetime TimeAsSeries[];
//--- установим доступ к массиву как к таймсерии
ArraySetAsSeries(TimeAsSeries,true);
ResetLastError();
int copied=CopyTime(NULL,0,0,10,TimeAsSeries);
if(copied<=0)
{
    Print("Не удалось скопировать время открытия для последних 10 баров");
    return;
}
Print("TimeCurrent = ",TimeCurrent());
Print("ArraySize(Time) = ",ArraySize(TimeAsSeries));
int size=ArraySize(TimeAsSeries);
for(int i=0;i<size;i++)
{
    Print("TimeAsSeries["+i+"] = ",TimeAsSeries[i]);
}

datetime ArrayNotSeries[];
ArraySetAsSeries(ArrayNotSeries,false);
ResetLastError();
copied=CopyTime(NULL,0,0,10,ArrayNotSeries);
if(copied<=0)
{
    Print("Не удалось скопировать время открытия для последних 10 баров");
    return;
}
size=ArraySize(ArrayNotSeries);
for(int i=size-1;i>=0;i--)
{

```

```
Print("ArrayNotSeries["+i+"] = ",ArrayNotSeries[i]);
}
```

В результате будет произведен вывод подобный этому:

```
TimeCurrent = 2009.06.11 14:16:23
ArraySize(Time) = 10
TimeAsSeries[0] = 2009.06.11 14:00:00
TimeAsSeries[1] = 2009.06.11 13:00:00
TimeAsSeries[2] = 2009.06.11 12:00:00
TimeAsSeries[3] = 2009.06.11 11:00:00
TimeAsSeries[4] = 2009.06.11 10:00:00
TimeAsSeries[5] = 2009.06.11 09:00:00
TimeAsSeries[6] = 2009.06.11 08:00:00
TimeAsSeries[7] = 2009.06.11 07:00:00
TimeAsSeries[8] = 2009.06.11 06:00:00
TimeAsSeries[9] = 2009.06.11 05:00:00

ArrayNotSeries[9] = 2009.06.11 14:00:00
ArrayNotSeries[8] = 2009.06.11 13:00:00
ArrayNotSeries[7] = 2009.06.11 12:00:00
ArrayNotSeries[6] = 2009.06.11 11:00:00
ArrayNotSeries[5] = 2009.06.11 10:00:00
ArrayNotSeries[4] = 2009.06.11 09:00:00
ArrayNotSeries[3] = 2009.06.11 08:00:00
ArrayNotSeries[2] = 2009.06.11 07:00:00
ArrayNotSeries[1] = 2009.06.11 06:00:00
ArrayNotSeries[0] = 2009.06.11 05:00:00
```

Как видно из результатов вывода, для массива TimeAsSeries с ростом индекса уменьшается значение времени, находящегося под этим индексом, то есть мы продвигаемся от настоящего к прошлому. Для обычного массива ArrayNotSeries все наоборот - с ростом индекса мы двигаемся из прошлого к настоящему.

#### Смотри также

[ArrayIsDynamic](#), [ArrayGetAsSeries](#), [ArraySetAsSeries](#), [ArrayIsSeries](#)

## Направление индексации в массивах, буферах и таймсерииях

Все массивы и индикаторные буфера по умолчанию имеют направление индексации слева направо. Индекс первого элемента всегда равен нулю. Таким образом, самый первый элемент массива или индикаторного буфера с индексом 0 по умолчанию находится на крайней левой позиции, последний элемент находится на крайней правой позиции.

Индикаторный буфер представляет из себя [динамический массив](#) типа `double`, размером которого управляет клиентский терминал с тем, чтобы он всегда соответствовал количеству баров, на которых индикатор рассчитывается. Обычный динамический массив типа `double` назначается в качестве индикаторного буфера с помощью функции [SetIndexBuffer\(\)](#). Для индикаторных буферов не требуется задавать размер с помощью функции [ArrayResize\(\)](#), исполняющая система терминала сама позаботится об этом.

[Таймсерии](#) представляют из себя массивы с обратной индексацией, то есть самый первый элемент таймсерии находится на крайней правой позиции, а последний элемент таймсерии находится на крайней левой позиции. Так как таймсерии предназначены для хранения исторических ценовых данных по финансовым инструментам и обязательно содержат информацию о времени, то можно сказать, что самые свежие данные в таймсерии находятся в правой крайней позиции, а самые старые в крайней левой позиции.

Поэтому, элемент с индексом ноль в таймсерии содержит информацию о самой последней котировке по инструменту. Если таймсерия представляет данные по дневному таймфрейму, то на нулевой позиции содержатся данные текущего незавершенного дня, а на позиции с индексом один хранятся данные вчерашнего дня.

### Изменение направления индексации

Функция [ArraySetAsSeries\(\)](#) позволяет изменять способ доступа к элементам динамического массива, но при этом физически порядок хранения данных в памяти компьютера не изменяется. Эта функция просто изменяет способ адресации к элементам массива, поэтому при копировании одного массива в другой с помощью функции [ArrayCopy\(\)](#) содержимое массива-приемника не будет зависеть от направления индексации в массиве-источнике.

Нельзя изменять направление индексации для статически распределенных массивов. Даже если массив был передан в качестве параметра в функцию, то и внутри этой функции попытки изменения направления индексации ни к чему не приведут.

Для индикаторных буферов, как и для обычных массивов, также разрешается устанавливать направление индексации задом наперед как в таймсерии, то есть, обращение к нулевой позиции в индикаторном буфере в этом случае будет означать обращение к самому последнему значению в соответствующем индикаторном буфере и это будет соответствовать значению индикатора на самом последнем баре. При этом физически размещение данных в индикаторном буфере останется неизменным, как уже упоминалось.

### Получение ценовых данных в индикаторах

В каждом [пользовательском индикаторе](#) обязательно должна присутствовать функция [OnCalculate\(\)](#), которой передаются ценовые данные, необходимые для расчета значений в

индикаторных буферах. Направление индексации в этих переданных массивах можно выяснить с помощью функции [ArrayGetAsSeries\(\)](#).

Переданные в функцию массивы отражают ценовые данные, т.е. эти массивы имеют признак таймсерии и функция [ArrayIsSeries\(\)](#) вернет true при проверке этих массивов. Но тем не менее, направление индексации необходимо в любом случае проверять только функцией [ArrayGetAsSeries\(\)](#).

Чтобы не зависеть от умолчаний, необходимо безусловно вызывать функцию [ArraySetAsSeries\(\)](#) для тех массивов, с которыми предполагается работать, и установить требуемое направление индексации.

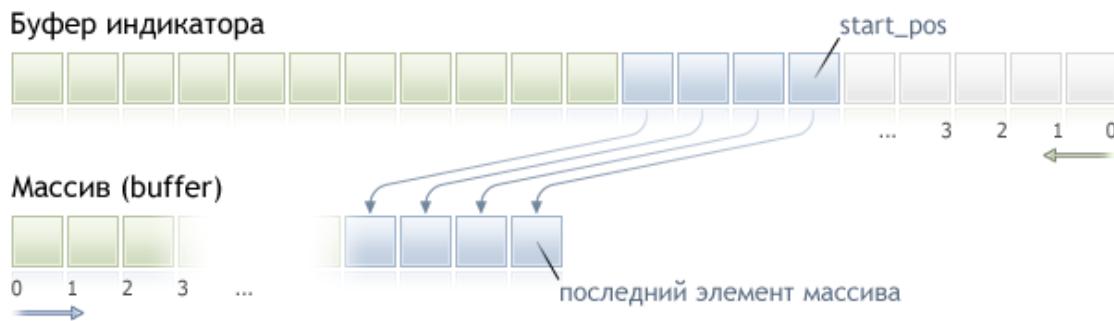
## Получение ценовых данных и значений индикаторов

В экспертах, индикаторах и скриптах все массивы по умолчанию имеют направление индексации слева направо. При необходимости в любой mq5-программе можно запросить значения таймсерий по любому символу и таймфрейму, а также значения индикаторов, рассчитанных на любом символе и таймфрейме.

Для получения таких данных служат функции Copy...():

- [CopyBuffer](#) - копирование значений индикаторного буфера в массив типа double;
- [CopyRates](#) - копирование ценовой истории в массив структур [MqlRates](#);
- [CopyTime](#) - копирование значений Time в массив типа datetime;
- [CopyOpen](#) - копирование значений Open в массив типа double;
- [CopyHigh](#) - копирование значений High в массив типа double;
- [CopyLow](#) - копирование значений Low в массив типа double;
- [CopyClose](#) - копирование значений Close в массив типа double;
- [CopyTickVolume](#) - копирование тиковых объемов в массив типа long;
- [CopyRealVolume](#) - копирование биржевых объемов в массив типа long;
- [CopySpread](#) - копирование истории спредов в массив типа int;

Все эти функции работают одинаково, и поэтому достаточно рассмотреть механизм получения данных на примере `CopyBuffer()`. Подразумевается, что все запрашиваемые данные имеют направление индексации, как в таймсерии, при этом подразумевается, что в позиции с индексом 0 (ноль) хранятся данные текущего незавершенного бара. Чтобы получить доступ к этим данным, нам необходимо скопировать нужный нам объем данных в массив-приемник, например, в массив `buffer`.



При копировании необходимо указать стартовую позицию в массиве-источнике, начиная с которой будут копироваться данные в массив-приемник. В случае успеха в массив-получатель будет скопировано указанное количество элементов из массива-источника, в данном случае из индикаторного буфера. При этом независимо от того, какое направление индексации установлено в массиве получателе, копирование всегда производится так, как указано на рисунке.

Если ценовые данные предполагается обрабатывать в цикле с большим количеством итераций, то рекомендуется проверять факт принудительного завершения программы с помощью функции [IsStopped\(\)](#):

```
int copied=CopyBuffer(ma_handle, // хэндл индикатора
                      0,          // индекс индикаторного буфера
                      0,          // стартовая позиция для копирования
                      number,    // количество значений для копирования
                      Buffer     // массив-получатель значений
                     );
if(copied<0) return;
int k=0;
while(k<copied && !IsStopped())
{
    //--- получаем значение для индекса k
    double value=Buffer[k];
    // ...
    // работа со значением value
    k++;
}
```

**Пример:**

```
input int per=10; // период экспоненты
int ma_handle;   // хэндл индикатора
//-----
//| Expert initialization function
//+-----+
int OnInit()
{
//---
ma_handle=iMA(_Symbol,0,per,0,MODE_EMA,PRICE_CLOSE);
//---
```

```
    return(INIT_SUCCEEDED);
}
//-----
//| Expert tick function
//+-----+
void OnTick()
{
//---
    double ema[10];
    int copied=CopyBuffer(ma_handle, // хэндл индикатора
                          0,          // индекс индикаторного буфера
                          0,          // стартовая позиция для копирования
                          10,         // количество значений для копирования
                          ema);       // массив-получатель значений
    );
    if(copied<0) return;
// .... дальнейший код
}
```

#### Смотри также

[Организация доступа к данным](#)

## Организация доступа к данным

В этом разделе рассматриваются вопросы, связанные с получением, хранением и запросами ценовых данных ([таймсериал](#)).

### Получение данных от торгового сервера

Прежде чем ценовые данные будут доступны в терминале MetaTrader 5, их необходимо получить и обработать. Для получения данных требуется подключение к торговому серверу MetaTrader 5. Данные поступают с сервера по запросу терминала в виде экономно упакованных блоков минутных баров.

Механизм обращения к серверу за данными не зависит от того, каким образом был инициирован запрос — пользователем при навигации по графику или программным способом на языке MQL5.

### Хранение промежуточных данных

Полученные с сервера данные автоматически распаковываются и сохраняются в специальном промежуточном формате НСС. Данные по каждому символу пишутся в отдельную папку `каталог_терминала\bases\имя_сервера\history\имя_символа`. Например, данные по символу EURUSD с торгового сервера MetaQuotes-Demo будут находиться в папке `каталог_терминала\bases\MetaQuotes-Demo\history\EURUSD\`.

Данные записываются в файлы с расширением .hcc, каждый файл хранит данные минутных баров за год. Например, файл 2009.hcc в папке EURUSD содержит минутные бары по символу EURUSD за 2009 год. Эти файлы используются для подготовки ценовых данных по всем таймфреймам и не предназначены для прямого доступа.

### Получение данных нужного таймфрейма из промежуточных данных

Служебные файлы в формате НСС исполняют роль источника данных для построения ценовых данных по запрошенным таймфреймам в формате НС. Данные в формате НС являются таймсериалами, максимально подготовленными для быстрого доступа. Они создаются только по запросу графика или mql5-программы в объеме, не превышающем значения параметра "Max bars in charts", и сохраняются для дальнейшего использования в файлах с расширением hc.

Для экономии ресурсов данные по таймфрейму загружаются и хранятся в оперативной памяти только по необходимости, при длительном отсутствии обращений к данным происходит выгрузка их из оперативной памяти с сохранением в файл. Для каждого таймфрейма данные подготавливаются независимо от наличия уже готовых данных для других таймфреймов. Правила формирования и доступности данных одинаковы для всех таймфреймов. Т.е. не смотря на то, что единицей хранения данных в формате НСС является минутный бар, наличие данных в формате НСС не означает наличие и доступность в том же объеме данных таймфрейма М1 в формате НС.

Получение новых данных с сервера вызывает автоматическое обновление используемых ценовых данных в формате НС по всем таймфреймам и перерасчет всех индикаторов, которые явно используют их в качестве входных данных для расчета.

### Параметр "Max bars in chart"

Параметр "Max bars in charts" ограничивает доступное для графиков, индикаторов и mql5-программ количество баров в формате НС. Это ограничение действует для данных всех таймфреймов, и предназначено в первую очередь для экономии ресурсов.

Устанавливая большие значения данного параметра, следует помнить, что при наличии достаточно глубокой истории ценовых данных для младших таймфреймов расход памяти на хранение таймсерий и буферов индикаторов может составить сотни мегабайт и достигнуть ограничения оперативной памяти для программы клиентского терминала (2Гб для 32-битных приложений MS Windows).

Изменение параметра "Max bars in charts" вступает в силу после перезапуска клиентского терминала. Само по себе изменение данного параметра не вызывает ни автоматического обращения к серверу за дополнительными данными, ни формирования дополнительных баров таймсерий. Запрос дополнительных ценовых данных у сервера и обновление таймсерий с учетом нового ограничения произойдет либо в случае прокрутки графика в область недостающих данных, либо в случае запроса недостающих данных из mql5-программы.

Объем запрашиваемых у сервера данных соответствует требуемому количеству баров данного таймфрейма с учетом значения параметра "Max bars in charts". Ограничение, задаваемое параметром, не является жестким, и в некоторых случаях количество доступных баров по таймфрейму может быть незначительно больше текущего значения параметра.

## Доступность данных

Наличие данных в формате НС или даже в готовом для использования формате НС не всегда означает безусловную доступность этих данных для отображения на графике или для использования в mql5-программах.

При доступе к ценовым данным или к значениям индикаторов из mql5-программ следует помнить, что не гарантируется их доступность в определенный момент времени, либо с определенного момента времени. Это связано с тем, что в целях экономии ресурсов в MetaTrader 5 не хранится полная копия требуемых данных для mql5-программы, а дается прямой доступ к базе данных терминала.

Ценовая история по всем таймфреймам строится из общих данных формата НС и любое обновление данных с сервера приводит к обновлению данных по всем таймфреймам и пересчету индикаторов. Вследствие этого, в доступе к данным может быть отказано даже в том случае, если эти данные были доступны мгновение назад.

## Синхронизация данных терминала и данных сервера

Поскольку mql5-программа может обратиться к данным по любому символу и таймфрейму, то есть вероятность, что данные требуемой таймсерии еще не сформированы в терминале или требуемые ценовые данные не синхронизированы с торговым сервером. В этом случае время ожидания готовности данных сложно прогнозировать.

Алгоритмы с использованием циклов ожидания готовности данных являются не лучшим решением. Единственное исключение в данном случае – скрипты, так как у них нет другого выбора алгоритма ввиду отсутствия обработки событий. Для пользовательских индикаторов подобные алгоритмы, как и любые другие циклы ожидания, категорически не рекомендуются, так как приводят к остановке расчета всех индикаторов и другой обработки ценовых данных по данному символу.

Для экспертов и пользовательских индикаторов лучше использовать [событийную модель обработки](#). Если при обработке события OnTick() или OnCalculate() не удалось получить все необходимые данные требуемой таймсерии, то следует выйти из обработчика события, рассчитывая на появление доступа к данным при следующем вызове обработчика.

## Пример скрипта для закачки истории

Рассмотрим пример скрипта, который выполняет запрос на получение истории с торгового сервера по указанному инструменту. Скрипт предназначен для запуска на графике требуемого инструмента, таймфрейм значения не имеет, так как, как уже было сказано выше, ценовые данные приходят с торгового сервера в виде упакованных минутных данных, из которых в последствие и строится любая предопределенная таймсериya.

Оформим все действия по получению данных в виде отдельной функции CheckLoadHistory(symbol, timeframe, start\_date):

```
int CheckLoadHistory(string symbol, ENUM_TIMEFRAMES period, datetime start_date)
{
}
```

Функция CheckLoadHistory() задумана как универсальная, которую могут вызвать из любой программы (эксперт, скрипт или индикатор), и поэтому требует три входных параметра: имя символа, период и начальную дату, с которой нам требуется ценовая история.

Вставим в код функции все необходимые проверки, прежде чем мы потребуем недостающую историю. Первым делом необходимо убедиться, что имя символа и значение периода являются корректными:

```
if(symbol==NULL || symbol=="") symbol=Symbol();
if(period==PERIOD_CURRENT) period=Period();
```

Далее – убедимся, что указанный символ доступен в окне MarketWatch, то есть, история по данному символу будет доступна при запросе к торговому серверу. Если его там нет – добавим символ в окно самостоятельно с помощью функции [SymbolSelect\(\)](#).

```
if(!SymbolInfoInteger(symbol, SYMBOL_SELECT))
{
    if(GetLastError()==ERR_MARKET_UNKNOWN_SYMBOL) return(-1);
    SymbolSelect(symbol,true);
}
```

Теперь необходимо получить начальную дату по уже имеющейся истории для указанной пары символ/период. Возможно, что значение входного параметра startdate, переданного функции CheckLoadHistory() попадает в интервал уже доступной истории, и тогда никакого запроса к торговому серверу не потребуется. Для получения самой первой даты по символу-периоду на данный момент предназначена функция [SeriesInfoInteger\(\)](#) с модификатором [SERIES\\_FIRSTDATE](#).

```
SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date);
if(first_date>0 && first_date<=start_date) return(1);
```

Следующая важная проверка – проверка типа программы, из которой вызывается функция. Напомним, что отправка запроса на обновление таймсерии с тем же периодом, что и у индикатора, вызывающего обновление, крайне нежелательна. Нежелательность запроса данных

по тому же символу-периоду, что и у индикатора обусловлена тем, что обновление исторических данных производится в том же потоке, в котором работает индикатор. Поэтому велика вероятность клинча. Для проверки используем функцию [MQL5InfoInteger\(\)](#) с модификатором [MQL5\\_PROGRAM\\_TYPE](#).

```
if(MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR && Period()==period && Syr
    return(-4);
```

Если мы успешно прошли все проверки, то сделаем последнюю попытку обойтись без обращения к торговому серверу. Сначала узнаем начальную дату, для которой доступны минутные данные в формате НСС. Запросим это значение функцией [SeriesInfoInteger\(\)](#) с модификатором [SERIES\\_TERMINAL\\_FIRSTDATE](#) и опять сравним со значением параметра `start_date`.

```
if(SeriesInfoInteger(symbol,PERIOD_M1,SERIES_TERMINAL_FIRSTDATE,first_date))
{
    //--- there is loaded data to build timeseries
    if(first_date>0)
    {
        //--- force timeseries build
        CopyTime(symbol,period,first_date+PeriodSeconds(period),1,times);
        //--- check date
        if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(2);
    }
}
```

Если после всех проверок поток выполнения по-прежнему находится в теле функции [CheckLoadHistory\(\)](#), то значит, есть необходимость запросить недостающие ценовые данные с торгового сервера. Для начала мы узнаем значение "Max bars in chart" с помощью функции [TerminalInfoInteger\(\)](#):

```
int max_bars=TerminalInfoInteger(TERMINAL_MAXBARS);
```

Оно нам понадобится для того, чтобы не запрашивать лишние данные. Затем выясним самую первую дату в истории по символу на торговом сервере (независимо от периода) посредством уже знакомой функции [SeriesInfoInteger\(\)](#) с модификатором [SERIES\\_SERVER\\_FIRSTDATE](#).

```
datetime first_server_date=0;
while(!SeriesInfoInteger(symbol,PERIOD_M1,SERIES_SERVER_FIRSTDATE,first_server_date))
    Sleep(5);
```

Так как запрос является асинхронной операцией, то функция вызывается в цикле с небольшой задержкой в 5 миллисекунд до тех пор, пока переменная `first_server_date` не получит значение либо выполнение цикла не будет прервано пользователем ([IsStopped\(\)](#) в этом случае вернет значение true). Укажем корректное значение начальной даты, начиная с которой мы запрашиваем у торгового сервера ценовые данные.

```
if(first_server_date>start_date) start_date=first_server_date;
if(first_date>0 && first_date<first_server_date)
    Print("Warning: first server date ",first_server_date," for ",symbol,
          " does not match to first series date ",first_date);
```

Если вдруг начальная дата *first\_server\_date* на сервере окажется меньше, чем начальная дата *first\_date* по символу в формате НСС , то в журнал будет выведено соответствующее сообщение.

Теперь мы готовы сделать запрос к торговому серверу за недостающими ценовыми данными. Запрос сделаем в виде цикла и начнем заполнять его тело:

```
while(!IsStopped())
{
    //1. дождаться синхронизации между перестроенной таймсерий и промежуточной исто
    //2. получить текущее количество баров bars на данной таймсерии
    //    если bars больше, чем Max_bars_in_chart, то можем выходить, работа окончен
    //3. Получим начальную дату first_date в перестроенной таймсерии и сравним со зв
    //    если first_date меньше чем start_date, то можем выходить, работа окончена
    //4. Запросим новую порцию истории в 100 баров у торгового сервера от последнег
}
```

Первые три пункта реализуются уже знакомыми приемами.

```
while(!IsStopped())
{
    //--- 1.дождаться окончания процесса перестройки таймсерии
    while(!SeriesInfoInteger(symbol,period,SERIES_SYNCHRONIZED) && !IsStopped())
        Sleep(5);

    //--- 2.запросим сколько баров мы теперь имеем
    int bars=Bars(symbol,period);
    if(bars>0)
    {
        //--- баров больше, чем можно отобразить на графике, выходим
        if(bars>=max_bars) return(-2);
        //--- 3. узнаем текущую начальную дату в таймсерии
        if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            // начальная дата более ранняя, чем запрашивалось, задача выполнена
            if(first_date>0 && first_date<=start_date) return(0);
    }
    //4. Запросим новую порцию истории в 100 баров у торгового сервера от последнег
}
```

Остался последний четвертый пункт – непосредственный запрос истории. Мы не можем прямо обратиться к серверу, но любая [Copy-функция](#) при нехватке истории в формате НСС терминал автоматически инициирует посылку такого запроса от терминала к торговому серверу. Так как время самой первой начальной даты в переменной *first\_date* является самым простым и естественным критерием для оценки степени выполнения запроса, то самым простым будет использовать функцию [CopyTime\(\)](#).

При вызове функций, осуществляющих копирование любых данных из таймсерий, необходимо иметь в виду то, что параметр *start* (номер бара, с которого начинать копирование ценовых данных) всегда должен быть в пределах доступной истории терминала. Если у нас имеется только 100 баров, то не имеет смысла пытаться скопировать 300 баров, начиная с бара с индексом 500. Такой запрос будет воспринят как ошибочный и не будет обработан, т.е. никакая история с торгового сервера подгружена не будет.

Именно поэтому мы будем копировать по 100 баров, начиная с бара с индексом bars. Это обеспечит плавную подкачку истории с торгового сервера, при этом реально будет подгружено чуть более запрошенных 100 баров, сервер отдает историю с запасом.

```
int copied=CopyTime(symbol,period,bars,100,times);
```

После операции копирования необходимо проанализировать количество скопированных элементов, если попытка оказалась неудачной, то значение переменной copied будет равно нулю и значение счетчика fail\_cnt будет увеличено на 1. После 100 неудачных попыток работа функции будет прервана.

```
int fail_cnt=0;
...
int copied=CopyTime(symbol,period,bars,100,times);
if(copied>0)
{
    //--- проверим данные
    if(times[0]<=start_date) return(0); // скопированное значение меньше, готово
    if(bars+copied>=max_bars) return(-2); // баров стало больше, чем помещается на экран
    fail_cnt=0;
}
else
{
    //--- не более 100 неудачных попыток подряд
    fail_cnt++;
    if(fail_cnt>=100) return(-5);
    Sleep(10);
}
```

Таким образом, в функции не только организована правильная обработка текущей ситуации в каждый момент выполнения, но и еще возвращается код завершения, который мы можем обработать после вызова функции CheckLoadHistory() для получения дополнительную информацию. Например, таким образом:

```
int res=CheckLoadHistory(InpLoadedSymbol,InpLoadedPeriod,InpStartDate);
switch(res)
{
    case -1 : Print("Неизвестный символ",InpLoadedSymbol);
    case -2 : Print("Запрошенных баров больше, чем можно отобразить на графике"); break;
    case -3 : Print("Выполнение было прервано пользователем");
    case -4 : Print("Индикатор не должен загружать собственные данные");
    case -5 : Print("Загрузка окончилась неудачей");
    case 0 : Print("Все данные загружены");
    case 1 : Print("Уже имеющихся данных в таймсерию достаточно");
    case 2 : Print("Таймсериya построена из имеющихся данных терминала");
    default : Print("Результат выполнения не определен");
}
```

Полный код функции приведен в примере скрипта, демонстрирующего правильный способ организации доступа к любым данным с обработкой результата запроса.

**Код:**

```

//+-----+
//| TestLoadHistory.mq5 |
//| Copyright 2009, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.02"
#property script_show_inputs
//--- input parameters
input string          InpLoadedSymbol="NZDUSD"; // Symbol to be load
input ENUM_TIMEFRAMES InpLoadedPeriod=PERIOD_H1; // Period to be load
input datetime         InpStartDate=D'2006.01.01'; // Start date
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    Print("Start load",InpLoadedSymbol+","+GetPeriodName(InpLoadedPeriod),"from",InpStart
//---
    int res=CheckLoadHistory(InpLoadedSymbol,InpLoadedPeriod,InpStartDate);
    switch(res)
    {
        case -1 : Print("Unknown symbol ",InpLoadedSymbol); break;
        case -2 : Print("Requested bars more than max bars in chart "); break;
        case -3 : Print("Program was stopped "); break;
        case -4 : Print("Indicator shouldn't load its own data "); break;
        case -5 : Print("Load failed "); break;
        case  0 : Print("Loaded OK "); break;
        case  1 : Print("Loaded previously "); break;
        case  2 : Print("Loaded previously and built "); break;
        default : Print("Unknown result ");
    }
//---
    datetime first_date;
    SeriesInfoInteger(InpLoadedSymbol,InpLoadedPeriod,SERIES_FIRSTDATE,first_date);
    int bars=Bars(InpLoadedSymbol,InpLoadedPeriod);
    Print("First date ",first_date," - ",bars," bars");
//---
}
//+-----+
//|
//+-----+
int CheckLoadHistory(string symbol,ENUM_TIMEFRAMES period,datetime start_date)
{
    datetime first_date=0;
    datetime times[100];

```

```

//--- check symbol & period
if(symbol==NULL || symbol=="") symbol=Symbol();
if(period==PERIOD_CURRENT) period=Period();
//--- check if symbol is selected in the MarketWatch
if(!SymbolInfoInteger(symbol,SYMBOL_SELECT))
{
    if(GetLastError()==ERR_MARKET_UNKNOWN_SYMBOL) return(-1);
    SymbolSelect(symbol,true);
}
//--- check if data is present
SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date);
if(first_date>0 && first_date<=start_date) return(1);
//--- don't ask for load of its own data if it is an indicator
if(MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR && Period()==period && Symbol()
return(-4);
//--- second attempt
if(SeriesInfoInteger(symbol,PERIOD_M1,SERIES_TERMINAL_FIRSTDATE,first_date))
{
    //--- there is loaded data to build timeseries
    if(first_date>0)
    {
        //--- force timeseries build
        CopyTime(symbol,period,first_date+PeriodSeconds(period),1,times);
        //--- check date
        if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(2);
    }
}
//--- max bars in chart from terminal options
int max_bars=TerminalInfoInteger(TERMINAL_MAXBARS);
//--- load symbol history info
datetime first_server_date=0;
while(!SeriesInfoInteger(symbol,PERIOD_M1,SERIES_SERVER_FIRSTDATE,first_server_date))
    Sleep(5);
//--- fix start date for loading
if(first_server_date>start_date) start_date=first_server_date;
if(first_date>0 && first_date<first_server_date)
    Print("Warning: first server date ",first_server_date," for ",symbol,
          " does not match to first series date ",first_date);
//--- load data step by step
int fail_cnt=0;
while(!IsStopped())
{
    //--- wait for timeseries build
    while(!SeriesInfoInteger(symbol,period,SERIES_SYNCHRONIZED) && !IsStopped())
        Sleep(5);
    //--- ask for built bars
    int bars=Bars(symbol,period);
    if(bars>0)

```

```

{
    if(bars>=max_bars) return(-2);
    //--- ask for first date
    if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
        if(first_date>0 && first_date<=start_date) return(0);
    }

    //--- copying of next part forces data loading
    int copied=CopyTime(symbol,period,bars,100,times);
    if(copied>0)
    {
        //--- check for data
        if(times[0]<=start_date) return(0);
        if(bars+copied>=max_bars) return(-2);
        fail_cnt=0;
    }
    else
    {
        //--- no more than 100 failed attempts
        fail_cnt++;
        if(fail_cnt>=100) return(-5);
        Sleep(10);
    }
}

//--- stopped
return(-3);
}

//+-----+
//| Возвращает строковое значение периода |
//+-----+
string GetPeriodName(ENUM_TIMEFRAMES period)
{
    if(period==PERIOD_CURRENT) period=Period();
    //---

    switch(period)
    {
        case PERIOD_M1: return("M1");
        case PERIOD_M2: return("M2");
        case PERIOD_M3: return("M3");
        case PERIOD_M4: return("M4");
        case PERIOD_M5: return("M5");
        case PERIOD_M6: return("M6");
        case PERIOD_M10: return("M10");
        case PERIOD_M12: return("M12");
        case PERIOD_M15: return("M15");
        case PERIOD_M20: return("M20");
        case PERIOD_M30: return("M30");
        case PERIOD_H1: return("H1");
        case PERIOD_H2: return("H2");
        case PERIOD_H3: return("H3");
    }
}

```

```
case PERIOD_H4: return("H4");
case PERIOD_H6: return("H6");
case PERIOD_H8: return("H8");
case PERIOD_H12: return("H12");
case PERIOD_D1: return("Daily");
case PERIOD_W1: return("Weekly");
case PERIOD_MN1: return("Monthly");
}
//---
return("unknown period");
}
```

## SeriesInfoInteger

Возвращает информацию о состоянии исторических данных. Существует 2 варианта функции.

**Непосредственно возвращает значение свойства.**

```
long SeriesInfoInteger(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    ENUM_SERIES_INFO_INTEGER prop_id, // идентификатор свойства
);
```

Возвращает true или false в зависимости от успешности выполнения функции.

```
bool SeriesInfoInteger(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    ENUM_SERIES_INFO_INTEGER prop_id, // идентификатор свойства
    long&           long_var        // переменная для получения информации
);
```

### Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*prop\_id*

[in] Идентификатор запрашиваемого свойства, значение перечисления [ENUM\\_SERIES\\_INFO\\_INTEGER](#).

*long\_var*

[out] Переменная, в которую помещается значение запрошенного свойства.

### Возвращаемое значение

Значение типа long для первого варианта вызова.

Для второго варианта вызова возвращает true, если данное свойство поддерживается и значение было помещено в переменную long\_var, иначе возвращает false. Чтобы получить дополнительную информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Пример:

```
void OnStart()
{
//---
Print("Количество баров по символу-периоду на данный момент = ",
      SeriesInfoInteger(Symbol(), Period(), SERIES_BARS_COUNT));
Print("Самая первая дата по символу-периоду на данный момент = ",
```

```
(datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_FIRSTDATE));  
  
Print("Самая первая дата в истории по символу на сервере = ",  
      (datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_SERVER_FIRSTDATE));  
  
Print("Данные по символу синхронизированы = ",  
      (bool)SeriesInfoInteger(Symbol(),Period(),SERIES_SYNCHRONIZED));  
}
```

## Bars

Возвращает количество баров в истории по соответствующему символу периоду. Существует 2 варианта функции.

**Запросить количество всех баров в истории**

```
int Bars(
    string      symbol_name,      // имя символа
    ENUM_TIMEFRAMES  timeframe   // периоду
);
```

**Запросить количество баров на заданном интервале**

```
int Bars(
    string      symbol_name,      // имя символа
    ENUM_TIMEFRAMES  timeframe,   // период
    datetime    start_time,       // с какой даты
    datetime    stop_time        // по какую дату
);
```

### Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

### Возвращаемое значение

Если указаны параметры *start\_time* и *stop\_time*, то функция возвращает количество баров в диапазоне дат. Если эти параметры не указаны, то функция возвращает общее количество баров.

### Примечание

Если данные для таймсерии с указанными параметрами при вызове функции Bars() еще не сформированы в терминале, или данные таймсерии в момент вызова функции не [синхронизированы](#) с торговым сервером, то функция вернет нулевое значение.

При запросе количества баров в заданном диапазоне дат учитываются только те бары, чье время открытия попадает в этот диапазон. Например, если текущий день недели – суббота, то при запросе количества недельных баров с указанием *start\_time=последний\_вторник* и *stop\_time=последняя\_пятница* функция вернет 0, так как время открытия на недельном таймфрейме всегда приходится на воскресенье, и ни один недельный бар не попадает в указанный диапазон.

Пример запроса количества всех баров в истории:

```

int bars=Bars(_Symbol,_Period);
if(bars>0)
{
    Print("Количество баров в истории терминала по символу-периоду на данный момент
}
else //нет доступных баров
{
    //--- видимо, данные по символу не синхронизированы с данными на сервере
    bool synchronized=false;
    //--- счетчик цикла
    int attempts=0;
    // сделаем 5 попыток дождаться синхронизации
    while(attempts<5)
    {
        if(SeriesInfoInteger(Symbol(),0,SERIES_SYNCHRONIZED))
        {
            //--- есть синхронизация, выходим
            synchronized=true;
            break;
        }
        //--- увеличим счетчик
        attempts++;
        //--- подождем 10 миллисекунд до следующей итерации
        Sleep(10);
    }
    //--- вышли из цикла по факту синхронизации
    if(synchronized)
    {
        Print("Количество баров в истории терминала по символу-периоду на данный момент
        Print("Самая первая в истории терминала дата по символу-периоду на данный момент =
            (datetime)SeriesInfoInteger(Symbol(),0,SERIES_FIRSTDATE));
        Print("Самая первая дата в истории по символу на сервере = ",
            (datetime)SeriesInfoInteger(Symbol(),0,SERIES_SERVER_FIRSTDATE));
    }
    //--- синхронизация данных так и не была достигнута
    else
    {
        Print("Не удалось получить количество баров на ",_Symbol);
    }
}

```

Пример запроса количества баров в заданном интервале:

```

int n;
datetime date1 = D'2016.09.02 23:55'; // пятница
datetime date2 = D'2016.09.05 00:00'; // понедельник
datetime date3 = D'2016.09.08 00:00'; // четверг
//---

```

```
n=Bars(_Symbol,PERIOD_H1,D'2016.09.02 02:05',D'2016.09.02 10:55');
Print("Количество баров: ",n); // Выведет "Количество баров: 8", в подсчете будет 9
n=Bars(_Symbol,PERIOD_D1,date1,date2);
Print("Количество баров: ",n); // Выведет "Количество баров: 1", поскольку в диапазоне
n=Bars(_Symbol,PERIOD_W1,date2,date3);
Print("Количество баров: ",n); // Выведет "Количество баров: 0", поскольку в заданном
```

**Смотри также**

[Функции обработки событий](#)

## BarsCalculated

Возвращает количество рассчитанных данных для запрашиваемого индикатора.

```
int BarsCalculated(
    int     indicator_handle,      // handle индикатора
);
```

### Параметры

*indicator\_handle*

[in] Хэндл индикатора, полученный соответствующей индикаторной функцией.

### Возвращаемое значение

Возвращает количество рассчитанных данных в индикаторном буфере или -1 в случае ошибки (данные еще не рассчитаны).

### Примечание

Функция полезна в тех случаях, когда необходимо получить данные индикатора сразу после его создания (получения хэндла индикатора).

### Пример:

```
void OnStart()
{
    double Ups[];
//--- установим для массивов признак таймсерии
    ArraySetAsSeries(Ups,true);
//--- создадим хэндл индикатора Fractals
    int FractalsHandle=iFractals(NULL,0);
//--- сбросим код ошибки
    ResetLastError();
//--- попытаемся скопировать значения индикатора
    int i,copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
    if(copied<=0)
    {
        Sleep(50);
        for(i=0;i<100;i++)
        {
            if(BarsCalculated(FractalsHandle)>0)
                break;
            Sleep(50);
        }
        copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
        if(copied<=0)
        {
            Print("Не удалось скопировать верхние фракталы. Error = ",GetLastError(),
            "i=",i,"    copied= ",copied);
            return;
        }
    }
}
```

```
        else
            Print("Удалось скопировать верхние фракталы.",
                  " i = ",i," copied = ",copied);
    }
else Print("Удалось скопировать верхние фракталы. ArraySize = ",ArraySize(Ups));
}
```

## IndicatorCreate

Возвращает хэндл указанного технического индикатора, созданного на основе массива параметров типа [MqlParam](#).

```
int IndicatorCreate(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    ENUM_INDICATOR  indicator_type,   // тип индикатора из перечисления
    int             parameters_cnt=0,  // количество параметров
    const MqlParam& parameters_array[]=NULL, // массив параметров
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*indicator\_type*

[in] Тип индикатора, может принимать одно из значений перечисления [ENUM\\_INDICATOR](#).

*parameters\_cnt*

[in] Количество параметров, передаваемых в массиве *parameters\_array[]*. Элементы массива имеют специальный тип структуры [MqlParam](#). По умолчанию нулевое значение - параметры не передаются. Если указано ненулевое количество параметров, то параметр *parameters\_array* является обязательным. Можно передавать не более 64 параметров.

*parameters\_array[]=NULL*

[in] Массив типа [MqlParam](#), элементы которого содержат тип и значение каждого входного параметра [технического индикатора](#).

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#).

### Примечание

Если создается хэндл индикатора типа IND\_CUSTOM, то поле *type* первого элемента массива входных параметров *parameters\_array* обязательно должен иметь значение TYPE\_STRING из перечисления [ENUM\\_DATATYPE](#), а поле *string\_value* первого элемента должно содержать имя пользовательского индикатора. Пользовательский индикатор должен быть скомпилирован (файл с расширением EX5) и находиться в директории MQL5/Indicators клиентского терминала или вложенной поддиректории.

Необходимые для тестирования индикаторы определяются автоматически из вызова функций *iCustom()*, если соответствующий параметр задан [константной строкой](#). Для остальных случаев

(использование функции [IndicatorCreate\(\)](#) или использование неконстантной строки в параметре, задающем имя индикатора) необходимо указать свойство [#property tester\\_indicator](#):

```
#property tester_indicator "indicator_name.ex5"
```

Если в пользовательском индикаторе используется [первая форма вызова](#), то при передаче входных параметров последним параметром можно дополнительно указать на каких данных он будет рассчитываться. Если параметр "Apply to" не указан явно, то по умолчанию расчет производится по значениям [PRICE\\_CLOSE](#).

**Пример:**

```
void OnStart()
{
    MqlParam params[];
    int      h_MA,h_MACD;
//--- create iMA("EURUSD",PERIOD_M15,8,0,MODE_EMA,PRICE_CLOSE);
    ArrayResize(params,4);
//--- set ma_period
    params[0].type      =TYPE_INT;
    params[0].integer_value=8;
//--- set ma_shift
    params[1].type      =TYPE_INT;
    params[1].integer_value=0;
//--- set ma_method
    params[2].type      =TYPE_INT;
    params[2].integer_value=MODE_EMA;
//--- set applied_price
    params[3].type      =TYPE_INT;
    params[3].integer_value=PRICE_CLOSE;
//--- create MA
    h_MA=IndicatorCreate("EURUSD",PERIOD_M15,IND_MA,4,params);
//--- create iMACD("EURUSD",PERIOD_M15,12,26,9,h_MA);
    ArrayResize(params,4);
//--- set fast ma_period
    params[0].type      =TYPE_INT;
    params[0].integer_value=12;
//--- set slow ma_period
    params[1].type      =TYPE_INT;
    params[1].integer_value=26;
//--- set smooth period for difference
    params[2].type      =TYPE_INT;
    params[2].integer_value=9;
//--- set indicator handle as applied_price
    params[3].type      =TYPE_INT;
    params[3].integer_value=h_MA;
//--- create MACD based on moving average
    h_MACD=IndicatorCreate("EURUSD",PERIOD_M15,IND_MACD,4,params);
//--- use indicators
```

```
//--- . . .
//--- release indicators (first h_MACD)
IndicatorRelease(h_MACD);
IndicatorRelease(h_MA);
}
```

## IndicatorParameters

Возвращает по указанному хэндулу количество входных параметров индикатора, а также сами значения и тип параметров.

```
int IndicatorParameters(
    int           indicator_handle,      // хэндл индикатора
    ENUM_INDICATOR& indicator_type,     // переменная для получения типа индикатора
    MqlParam&       parameters[]        // массив для получения параметров
);
```

### Параметры

*indicator\_handle*

[in] Хэндл индикатора, для которого необходимо узнать количество параметров, на которых он рассчитан.

*indicator\_type*

[out] Переменная типа [ENUM\\_INDICATOR](#), в которую будет записан тип индикатора.

*parameters[]*

[out] Динамический массив для получения значений типа [MqlParam](#), в который будет записан список параметров индикатора. Размер массива возвращает сама функция `IndicatorParameters()`.

### Возвращаемое значение

Количество входных параметров индикатора с указанным хэндлом, в случае ошибки возвращает -1. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    //--- количество окон на графике (всегда есть хотя бы одно главное окно)
    int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
    //--- проходим по окнам графика
    for(int w=0;w<windows;w++)
    {
        //--- сколько индикаторов в данном окне/подконе
        int total=ChartIndicatorsTotal(0,w);
        //--- переберем все индикаторы в окне
        for(int i=0;i<total;i++)
        {
            //--- получим короткое имя индикатора
            string name=ChartIndicatorName(0,w,i);
            //--- получим хэндл индикатора
            int handle=ChartIndicatorGet(0,w,name);
```

```
//--- выводим в журнал
PrintFormat("Window=%d, indicator #%d, handle=%d",w,i,handle);
//---

MqlParam parameters[];
ENUM_INDICATOR indicator_type;
int params=IndicatorParameters(handle,indicator_type,parameters);
//--- заголовок сообщения
string par_info="Short name "+name+", type "
+EnumToString(ENUM_INDICATOR(indicator_type))+"\r\n";
//---

for(int p=0;p<params;p++)
{
    par_info+=StringFormat("parameter %d: type=%s, long_value=%d, double_value=%f",
                           p,
                           EnumToString((ENUM_DATATYPE)parameters[p].type),
                           parameters[p].integer_value,
                           parameters[p].double_value,
                           parameters[p].string_value
                           );
}
Print(par_info);
}

//--- прошли по всем индикаторам в окне
}

//---
}
```

Смотри также

[ChartIndicatorGet\(\)](#)

## IndicatorRelease

Удаляет хэндл индикатора и освобождает расчетную часть индикатора, если ею больше никто не пользуется.

```
bool IndicatorRelease(
    int indicator_handle // handle индикатора
);
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Функция позволяет удалять хэндл индикатора, если он больше не нужен, и таким образом позволяет экономить память. Удаление хендла производится сразу, удаление расчетной части индикатора производится через некоторое небольшое время (если обращений к ней больше нет).

При работе в [тестере стратегий](#) функция IndicatorRelease() не выполняется.

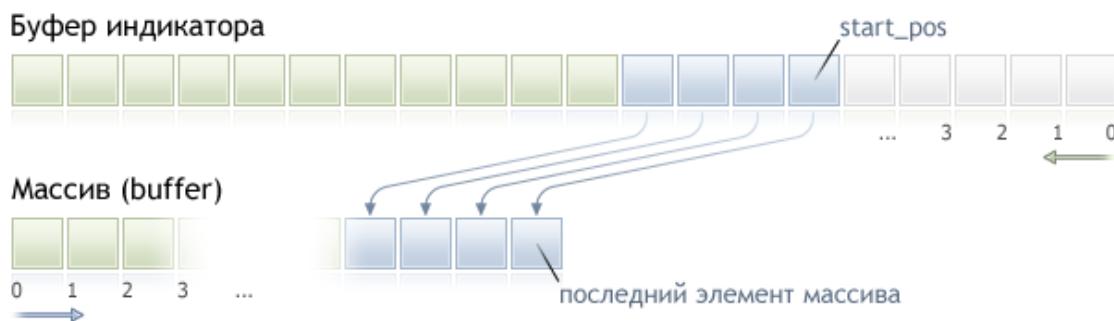
### Пример:

```
//-----+
//|                               Test_IndicatorRelease.mq5 |
//|                               Copyright 2010, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
input int          MA_Period=15;
input int          MA_shift=0;
input ENUM_MA_METHOD MA_smooth=MODE_SMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
//--- будем хранить хендл индикатора
int MA_handle=INVALID_HANDLE;
//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//--- создадим хендл индикатора
MA_handle=iMA(Symbol(),0,MA_Period,MA_shift,MA_smooth,PRICE_CLOSE);
//--- удаление глобальной переменной
if(GlobalVariableCheck("MA_value"))
    GlobalVariableDel("MA_value");
//---
return(INIT_SUCCEEDED);
}
```

```
//+-----+
//| Expert tick function
//+-----+
void OnTick()
{
//--- если глобальной переменной еще нет
if(!GlobalVariableCheck("MA_value"))
{
//--- получим значения индикатора на двух последних барах
if(MA_handle!=INVALID_HANDLE)
{
//--- динамический массив для получения значений индикатора
double values[];
if(CopyBuffer(MA_handle,0,0,2,values)==2 && values[0]!=EMPTY_VALUE)
{
//--- запомним в глобальной переменной значение на предпоследнем баре
if(GlobalVariableSet("MA_value",values[0]))
{
//--- освободим хэндл индикатора
if(!IndicatorRelease(MA_handle))
Print("IndicatorRelease() failed. Error ",GetLastError());
else MA_handle=INVALID_HANDLE;
}
else
Print("GlobalVariableSet failed. Error ",GetLastError());
}
}
}
//---
}
```

## CopyBuffer

Получает в массив buffer данные указанного буфера указанного индикатора в указанном количестве.



Отсчет элементов копируемых данных (индикаторный буфер с индексом buffer\_num) от стартовой позиции ведется от настоящего к прошлому, то есть стартовая позиция, равная 0, означает текущий бар (значение индикатора для текущего бара).

При копировании заранее неизвестного количества данных в качестве массива-приемника buffer[] желательно использовать [динамический массив](#), так как функция CopyBuffer() старается распределить размер принимающего массива под размер копируемых данных. Если в качестве принимающего массива buffer[] выступает индикаторный буфер (массив, предварительно назначенный под хранение значений индикатора функцией [SetIndexBuffer\(\)](#)), то допускается частичное копирование. Пример можно посмотреть в пользовательском индикаторе Awesome\_Oscillator.mq5 из стандартной поставки терминала.

Если необходимо произвести частичное копирование значений индикатора в другой массив (не индикаторный буфер), то для этих целей необходимо использовать промежуточный массив, в который копируется требуемое количество. И уже из этого массива-посредника произвести поэлементное копирование нужного количества значений в нужные места принимающего массива.

Если необходимо копировать заранее известное количество данных, то лучше это делать в [статически выделенный буфер](#), чтобы избежать излишнего перевыделения памяти.

Неважно, какое свойство имеет приемный массив - as\_series=true или as\_series=false, данные будут скопированы таким образом, что самый старый по времени элемент будет в начале физической памяти, отведенной под массив. Существует 3 варианта функции.

### Обращение по начальной позиции и количеству требуемых элементов

```
int CopyBuffer(
    int      indicator_handle,      // handle индикатора
    int      buffer_num,           // номер буфера индикатора
    int      start_pos,            // откуда начнем
    int      count,                // сколько копируем
    double   buffer[]             // массив, куда будут скопированы данные
);
```

### Обращение по начальной дате и количеству требуемых элементов

```

int CopyBuffer(
    int      indicator_handle,      // handle индикатора
    int      buffer_num,           // номер буфера индикатора
    datetime start_time,          // с какой даты
    int      count,                // сколько копируем
    double   buffer[]             // массив, куда будут скопированы данные
);

```

### Обращение по начальной и конечной датам требуемого интервала времени

```

int CopyBuffer(
    int      indicator_handle,      // handle индикатора
    int      buffer_num,           // номер буфера индикатора
    datetime start_time,          // с какой даты
    datetime stop_time,           // по какую дату
    double   buffer[]             // массив, куда будут скопированы данные
);

```

#### Параметры

*indicator\_handle*

[in] Хэндл индикатора, полученный соответствующей индикаторной функцией.

*buffer\_num*

[in] Номер индикаторного буфера.

*start\_pos*

[in] Номер первого копируемого элемента.

*count*

[in] Количество копируемых элементов.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

*buffer[]*

[out] Массив типа [double](#).

#### Возвращаемое значение

Количество скопированных элементов массива либо -1 в случае [ошибки](#).

#### Примечание

При запросе данных из индикатора, если запрашиваемые таймсерии еще не построены или их необходимо загрузить с сервера, то функция сразу же вернет -1, но при этом сам процесс загрузки/построения будет инициирован.

При запросе данных из эксперта или скрипта, будет инициирована [загрузка с сервера](#), если локально этих данных у терминала нет, либо начнется построение нужной таймсерии, если

данные можно построить из локальной истории, но они еще не готовы. Функция вернет то количество данных, которые будут готовы к моменту истечения таймаута.

### Пример:

```

//+-----+
//| TestCopyBuffer3.mq5 |
//| Copyright 2009, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot MA
#property indicator_label1 "MA"
#property indicator_type1  DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameters
input bool           AsSeries=true;
input int            period=15;
input ENUM_MA_METHOD smootMode=MODE_EMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
input int            shift=0;
//--- indicator buffers
double              MABuffer[];
int                 ma_handle;
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
    Print("Параметр AsSeries = ",AsSeries);
    Print("Индикаторный буфер после SetIndexBuffer() является таймсерий = ",
          ArrayGetAsSeries(MABuffer));
//--- set short indicator name
    IndicatorSetString(INDICATOR_SHORTNAME,"MA("+period+"")"+AsSeries);
//--- set AsSeries (depends on input parameter)
    ArraySetAsSeries(MABuffer,AsSeries);
    Print("Индикаторный буфер после ArraySetAsSeries(MABuffer,true); является таймсерий =
          ArrayGetAsSeries(MABuffer));
//---
    ma_handle=iMA(Symbol(),0,period,shift,smootMode,price);
}

```

```

        return(INIT_SUCCEEDED);
    }
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- check if all data calculated
    if(BarsCalculated(ma_handle)<rates_total) return(0);
//--- we can copy not all data
    int to_copy;
    if(prev_calculated>rates_total || prev_calculated<=0) to_copy=rates_total;
    else
    {
        to_copy=rates_total-prev_calculated;
        //--- last value is always copied
        to_copy++;
    }
//--- try to copy
    if(CopyBuffer(ma_handle,0,0,to_copy,MABuffer)<=0) return(0);
//--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+

```

В вышеприведенном примере проиллюстрировано заполнение индикаторного буфера значениями другого индикаторного буфера от индикатора на том же символе/периоде.

Более полный пример запроса исторических данных смотрите в разделе [Способы привязки объектов](#). В приведенном там скрипте показано, как получать значения индикатора [iFractals](#) на последних 1000 барах и как потом вывести на график по десять последних фракталов вверх и вниз. Подобный прием можно использовать для всех индикаторов, которые имеют пропуски значений и обычно реализуются с помощью следующих [стилей построения](#):

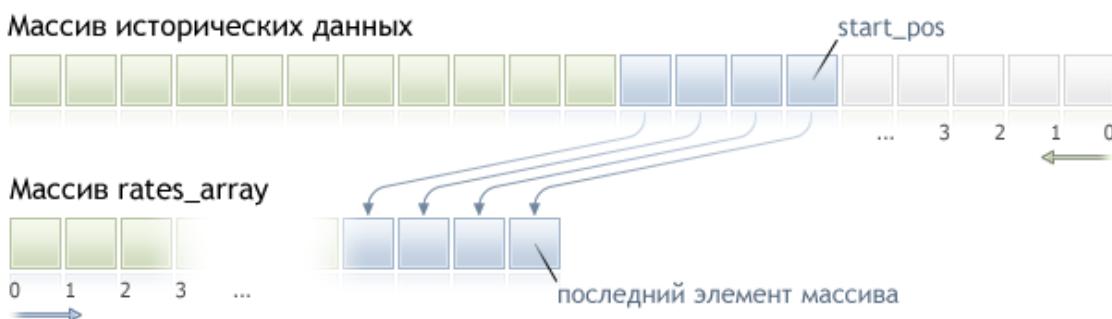
- [DRAW\\_SECTION](#),
- [DRAW\\_ARROW](#),
- [DRAW\\_ZIGZAG](#),
- [DRAW\\_COLOR\\_SECTION](#),
- [DRAW\\_COLOR\\_ARROW](#),
- [DRAW\\_COLOR\\_ZIGZAG](#).

Смотри также

[Свойства пользовательских индикаторов, SetIndexBuffer](#)

## CopyRates

Получает в массив rates\_array исторические данные структуры [MqlRates](#) указанного символа-периода в указанном количестве. Отсчет элементов от стартовой позиции ведется от настоящего к прошлому, то есть стартовая позиция, равная 0, означает текущий бар.



При копировании заранее неизвестного количества данных рекомендуется в качестве приемного массива использовать [динамический массив](#), так как если данных оказывается меньше (или больше), чем вмещает массив, то производится попытка перераспределения массива таким образом, чтобы запрошенные данные поместились целиком и полностью.

Если необходимо копировать заранее известное количество данных, то лучше это делать в [статически выделенный буфер](#), чтобы избежать излишнего перевыделения памяти.

Неважно, какое свойство имеет приемный массив - as\_series=true или as\_series=false, данные будут скопированы таким образом, что самый старый по времени элемент будет в начале физической памяти, отведенной под массив. Существует 3 варианта функции.

### Обращение по начальной позиции и количеству требуемых элементов

```
int CopyRates(
    string          symbol_name,           // имя символа
    ENUM_TIMEFRAMES timeframe,            // период
    int             start_pos,              // откуда начнем
    int             count,                  // сколько копируем
    MqlRates        rates_array[]         // массив, куда будут скопированы данные
);
```

### Обращение по начальной дате и количеству требуемых элементов

```
int CopyRates(
    string          symbol_name,           // имя символа
    ENUM_TIMEFRAMES timeframe,            // период
    datetime        start_time,             // с какой даты
    int             count,                  // сколько копируем
    MqlRates        rates_array[]         // массив, куда будут скопированы данные
);
```

### Обращение по начальной и конечной датам требуемого интервала времени

```
int CopyRates(
```

```

string          symbol_name,           // имя символа
ENUM_TIMEFRAMES timeframe,          // период
datetime        start_time,          // с какой даты
datetime        stop_time,           // по какую дату
MqlRates        rates_array[];      // массив, куда будут скопированы данные
);

```

## Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*start\_pos*

[in] Номер первого копируемого элемента.

*count*

[in] Количество копируемых элементов.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

*rates\_array[]*

[out] Массив типа [MqlRates](#).

## Возвращаемое значение

Количество скопированных элементов массива либо -1 в случае [ошибки](#).

## Примечание

Если интервал запрашиваемых данных полностью находится вне доступных данных на сервере, то функция возвращает -1. В случае если запрашиваются данные за пределами [TERMINAL\\_MAXBARS](#) (максимальное количество баров на графике), функция также вернет -1.

При запросе данных из индикатора, если запрашиваемые таймсерии еще не построены или их необходимо загрузить с сервера, то функция сразу же вернет -1, но при этом сам процесс загрузки/построения будет инициирован.

При запросе данных из эксперта или скрипта, будет инициирована [загрузка с сервера](#), если локально этих данных у терминала нет, либо начнется построение нужной таймсерии, если данные можно построить из локальной истории, но они еще не готовы. Функция вернет то количество данных, которые будут готовы к моменту истечения таймаута, но загрузка истории будет продолжаться, и при следующем аналогичном запросе функция вернет уже больше данных.

При запросе данных по начальной дате и количеству требуемых элементов возвращаются только данные, дата которых меньше (раньше) или равна указанной. При этом интервал задается и учитывается с точностью до секунды. То есть дата открытия любого бара, для которого

возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда равна или меньше указанной.

При запросе данных в заданном диапазоне дат возвращаются только данные, попадающие в запрашиваемый интервал, при этом интервал задается и учитывается с точностью до секунды. То есть время открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда находится в запрошенном интервале.

Таким образом, если текущий день недели Суббота, то при попытке скопировать данные на недельном таймфрейме с указанием `start_time=Последний_Вторник` и `stop_time=Последняя_Пятница` функция вернет 0, так как время открытия на недельном таймфрейме всегда приходится на воскресенье, но ни один недельный бар не попадает в указанный диапазон.

Если необходимо получить значение, соответствующее текущему незавершенному бару, то можно использовать первую форму вызова с указанием `start_pos=0` и `count=1`.

**Пример:**

```
void OnStart()
{
//---

    MqlRates rates[];
    ArraySetAsSeries(rates,true);
    int copied=CopyRates(Symbol(),0,0,100,rates);
    if(copied>0)
    {
        Print("Скопировано баров: "+copied);
        string format="open = %G, high = %G, low = %G, close = %G, volume = %d";
        string out;
        int size=fmin(copied,10);
        for(int i=0;i<size;i++)
        {
            out=i+":"+TimeToString(rates[i].time);
            out=out+" "+StringFormat(format,
                rates[i].open,
                rates[i].high,
                rates[i].low,
                rates[i].close,
                rates[i].tick_volume);
            Print(out);
        }
    }
    else Print("Не удалось получить исторические данные по символу ",Symbol());
}
```

Более полный пример запроса исторических данных смотрите в разделе [Способы привязки объектов](#). В приведенном там скрипте показано, как получать значения индикатора [iFractals](#) на последних 1000 барах и как потом вывести на график по десять последних фракталов вверх и

вниз. Подобный прием можно использовать для всех индикаторов, которые имеют пропуски значений и обычно реализуются с помощью следующих [стилей построения](#):

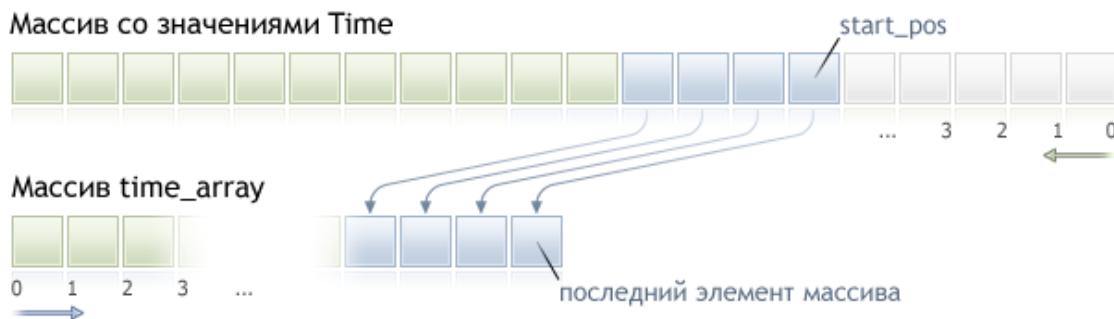
- [DRAW\\_SECTION](#),
- [DRAW\\_ARROW](#),
- [DRAW\\_ZIGZAG](#),
- [DRAW\\_COLOR\\_SECTION](#),
- [DRAW\\_COLOR\\_ARROW](#),
- [DRAW\\_COLOR\\_ZIGZAG](#).

Смотри также

[Структуры и классы](#), [TimeToString](#), [StringFormat](#)

## CopyTime

Функция получает в массив time\_array исторические данные времени открытия баров для указанной пары символ-период в указанном количестве. Необходимо отметить, что отсчет элементов от стартовой позиции ведется от настоящего к прошлому, то есть стартовая позиция, равная 0, означает текущий бар.



При копировании заранее неизвестного количества данных рекомендуется в качестве приемного массива использовать [динамический массив](#), так как если данных оказывается меньше (или больше), чем вмещает массив, то производится попытка перераспределения массива таким образом, чтобы запрошенные данные поместились целиком и полностью.

Если необходимо копировать заранее известное количество данных, то лучше это делать в [статически выделенный буфер](#), чтобы избежать излишнего перевыделения памяти.

Неважно, какое свойство имеет приемный массив - as\_series=true или as\_series=false, данные будут скопированы таким образом, что самый старый по времени элемент будет в начале физической памяти, отведенной под массив. Существует 3 варианта функции.

### Обращение по начальной позиции и количеству требуемых элементов

```
int CopyTime(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    int             start_pos,        // откуда начнем
    int             count,            // сколько копируем
    datetime        time_array[]     // массив для копирования времени открытия
);
```

### Обращение по начальной дате и количеству требуемых элементов

```
int CopyTime(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,        // с какой даты
    int             count,            // сколько копируем
    datetime        time_array[]     // массив для копирования времени открытия
);
```

### Обращение по начальной и конечной датам требуемого интервала времени

```
int CopyTime(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,       // с какой даты
    datetime        stop_time,        // по какую дату
    datetime        time_array[]     // массив для копирования времени открытия
);
```

## Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*start\_pos*

[in] Номер первого копируемого элемента.

*count*

[in] Количество копируемых элементов.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

*time\_array[]*

[out] Массив типа [datetime](#).

## Возвращаемое значение

Количество скопированных элементов массива либо -1 в случае [ошибки](#).

## Примечание

Если интервал запрашиваемых данных полностью находится вне доступных данных на сервере, то функция возвращает -1. В случае если запрашиваются данные за пределами [TERMINAL\\_MAXBARS](#) (максимальное количество баров на графике), функция также вернет -1.

При запросе данных из индикатора, если запрашиваемые таймсерии еще не построены или их необходимо загрузить с сервера, функция сразу же вернет -1, но при этом сам процесс загрузки/построения будет инициирован.

При запросе данных из эксперта или скрипта, будет иницирована [загрузка с сервера](#), если локально этих данных у терминала нет, либо начнется построение нужной таймсерии, если данные можно построить из локальной истории, но они еще не готовы. Функция вернет то количество данных, которые будут готовы к моменту истечения таймаута, но загрузка истории будет продолжаться, и при следующем аналогичном запросе функция вернет уже больше данных.

При запросе данных по начальной дате и количеству требуемых элементов возвращаются только данные, дата которых меньше (раньше) или равна указанной. При этом интервал задается и учитывается с точностью до секунды. То есть дата открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда равна или меньше указанной.

При запросе данных в заданном диапазоне дат возвращаются только данные, попадающие в запрашиваемый интервал, при этом интервал задается и учитывается с точностью до секунды. То есть время открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда находится в запрошенном интервале.

Таким образом, если текущий день недели Суббота, то при попытке скопировать данные на недельном таймфрейме с указанием `start_time=Последний_Вторник` и `stop_time=Последняя_Пятница` функция вернет 0, так как время открытия на недельном таймфрейме всегда приходится на воскресенье, но ни один недельный бар не попадает в указанный диапазон.

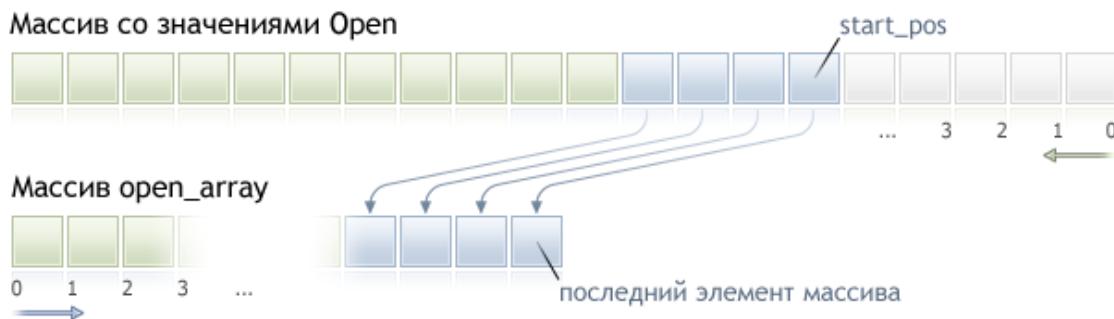
Если необходимо получить значение, соответствующее текущему незавершенному бару, то можно использовать первую форму вызова с указанием `start_pos=0` и `count=1`.

Пример запроса исторических данных смотрите в разделе [Способы привязки объектов](#). В приведенном там скрипте показано, как получать значения индикатора `iFractals` на последних 1000 барах и как потом вывести на график по десять последних фракталов вверх и вниз. Подобный прием можно использовать для всех индикаторов, которые имеют пропуски значений и обычно реализуются с помощью следующих [стилей построения](#):

- [DRAW\\_SECTION](#),
- [DRAW\\_ARROW](#),
- [DRAW\\_ZIGZAG](#),
- [DRAW\\_COLOR\\_SECTION](#),
- [DRAW\\_COLOR\\_ARROW](#),
- [DRAW\\_COLOR\\_ZIGZAG](#).

## CopyOpen

Функция получает в массив open\_array исторические данные цен открытия баров для указанной пары символ-период в указанном количестве. Необходимо отметить, что отсчет элементов от стартовой позиции ведется от настоящего к прошлому, то есть стартовая позиция, равная 0, означает текущий бар.



При копировании заранее неизвестного количества данных рекомендуется в качестве приемного массива использовать [динамический массив](#), так как если данных оказывается меньше (или больше), чем вмещает массив, то производится попытка перераспределения массива таким образом, чтобы запрошенные данные поместились целиком и полностью.

Если необходимо копировать заранее известное количество данных, то лучше это делать в [статически выделенный буфер](#), чтобы избежать излишнего перевыделения памяти.

Неважно, какое свойство имеет приёмный массив - as\_series=true или as\_series=false, данные будут скопированы таким образом, что самый старый по времени элемент будет в начале физической памяти, отведенной под массив. Существует 3 варианта функции.

### Обращение по начальной позиции и количеству требуемых элементов

```
int CopyOpen(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    int             start_pos,        // откуда начнем
    int             count,            // сколько копируем
    double          open_array[]     // массив для копирования цен открытия
);
```

### Обращение по начальной дате и количеству требуемых элементов

```
int CopyOpen(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,        // с какой даты
    int             count,            // сколько копируем
    double          open_array[]     // массив для копирования цен открытия
);
```

### Обращение по начальной и конечной датам требуемого интервала времени

```

int CopyOpen(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,       // с какой даты
    datetime        stop_time,        // по какую дату
    double          open_array[]     // массив для копирования цен открытия
);

```

### Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*start\_pos*

[in] Номер первого копируемого элемента.

*count*

[in] Количество копируемых элементов.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

*open\_array[]*

[out] Массив типа [double](#).

### Возвращаемое значение

Количество скопированных элементов массива либо -1 в случае [ошибки](#).

### Примечание

Если интервал запрашиваемых данных полностью находится вне доступных данных на сервере, то функция возвращает -1. В случае если запрашиваются данные за пределами [TERMINAL\\_MAXBARS](#) (максимальное количество баров на графике), функция также вернет -1.

При запросе данных из индикатора, если запрашиваемые таймсерии еще не построены или их необходимо загрузить с сервера, функция сразу же вернет -1, но при этом сам процесс загрузки/построения будет инициирован.

При запросе данных из эксперта или скрипта, будет инициирована [загрузка с сервера](#), если локально этих данных у терминала нет, либо начнется построение нужной таймсерии, если данные можно построить из локальной истории, но они еще не готовы. Функция вернет то количество данных, которые будут готовы к моменту истечения таймаута, но загрузка истории будет продолжаться, и при следующем аналогичном запросе функция вернет уже больше данных.

При запросе данных по начальной дате и количеству требуемых элементов возвращаются только данные, дата которых меньше (раньше) или равна указанной. При этом интервал задается и

учитывается с точностью до секунды. То есть дата открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда равна или меньше указанной.

При запросе данных в заданном диапазоне дат возвращаются только данные, попадающие в запрашиваемый интервал, при этом интервал задается и учитывается с точностью до секунды. То есть время открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда находится в запрошенном интервале.

Таким образом, если текущий день недели Суббота, то при попытке скопировать данные на недельном таймфрейме с указанием `start_time=Последний_Вторник` и `stop_time=Последняя_Пятница` функция вернет 0, так как время открытия на недельном таймфрейме всегда приходится на воскресенье, но ни один недельный бар не попадает в указанный диапазон.

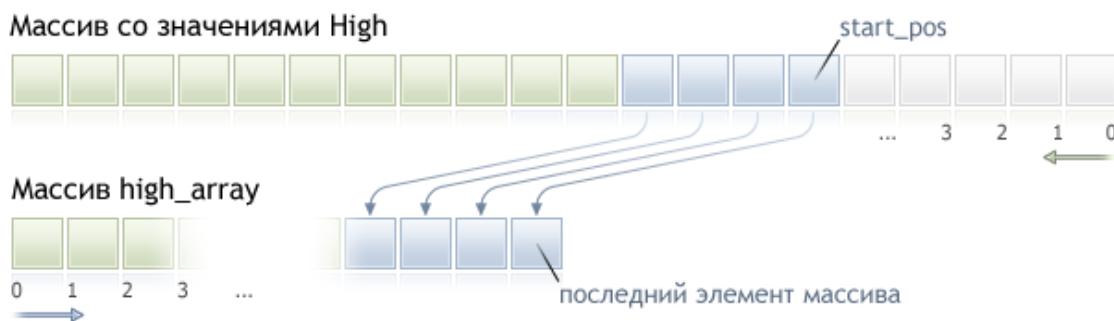
Если необходимо получить значение, соответствующее текущему незавершенному бару, то можно использовать первую форму вызова с указанием `start_pos=0` и `count=1`.

Пример запроса исторических данных смотрите в разделе [Способы привязки объектов](#). В приведенном там скрипте показано, как получать значения индикатора `iFractals` на последних 1000 барах и как потом вывести на график по десять последних фракталов вверх и вниз. Подобный прием можно использовать для всех индикаторов, которые имеют пропуски значений и обычно реализуются с помощью следующих [стилей построения](#):

- [DRAW\\_SECTION](#),
- [DRAW\\_ARROW](#),
- [DRAW\\_ZIGZAG](#),
- [DRAW\\_COLOR\\_SECTION](#),
- [DRAW\\_COLOR\\_ARROW](#),
- [DRAW\\_COLOR\\_ZIGZAG](#).

## CopyHigh

Функция получает в массив `high_array` исторические данные максимальных цен баров для указанной пары символ-период в указанном количестве. Необходимо отметить, что отсчет элементов от стартовой позиции ведется от настоящего к прошлому, то есть стартовая позиция, равная 0, означает текущий бар.



При копировании заранее неизвестного количества данных рекомендуется в качестве приемного массива использовать [динамический массив](#), так как если данных оказывается меньше (или больше), чем вмещает массив, то производится попытка перераспределения массива таким образом, чтобы запрошенные данные поместились целиком и полностью.

Если необходимо копировать заранее известное количество данных, то лучше это делать в [статически выделенный буфер](#), чтобы избежать излишнего перевыделения памяти.

Неважно, какое свойство имеет приемный массив - `as_series=true` или `as_series=false`, данные будут скопированы таким образом, что самый старый по времени элемент будет в начале физической памяти, отведенной под массив. Существует 3 варианта функции.

### Обращение по начальной позиции и количеству требуемых элементов

```
int CopyHigh(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    int             start_pos,        // откуда начнем
    int             count,            // сколько копируем
    double          high_array[]     // массив для копирования максимальных цен
);
```

### Обращение по начальной дате и количеству требуемых элементов

```
int CopyHigh(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,        // с какой даты
    int             count,            // сколько копируем
    double          high_array[]     // массив для копирования максимальных цен
);
```

### Обращение по начальной и конечной датам требуемого интервала времени

```

int CopyHigh(
    string          symbol_name,           // имя символа
    ENUM_TIMEFRAMES timeframe,           // период
    datetime        start_time,           // с какой даты
    datetime        stop_time,            // по какую дату
    double          high_array[]         // массив для копирования максимальных цен
);

```

### Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*start\_pos*

[in] Номер первого копируемого элемента.

*count*

[in] Количество копируемых элементов.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

*high\_array[]*

[out] Массив типа [double](#).

### Возвращаемое значение

Количество скопированных элементов массива либо -1 в случае [ошибки](#).

### Примечание

Если интервал запрашиваемых данных полностью находится вне доступных данных на сервере, то функция возвращает -1. В случае, если запрашиваются данные за пределами [TERMINAL\\_MAXBARS](#) (максимальное количество баров на графике), то функция также вернет -1.

При запросе данных из индикатора, если запрашиваемые таймсерии еще не построены или их необходимо загрузить с сервера, то функция сразу же вернет -1, но при этом сам процесс загрузки/построения будет инициирован.

При запросе данных из эксперта или скрипта, будет инициирована [загрузка с сервера](#), если локально этих данных у терминала нет, либо начнется построение нужной таймсерии, если данные можно построить из локальной истории, но они еще не готовы. Функция вернет то количество данных, которые будут готовы к моменту истечения таймаута, но загрузка истории будет продолжаться и при следующем аналогичном запросе функция вернет уже больше данных.

При запросе данных по начальной дате и количеству требуемых элементов возвращаются только данные, дата которых меньше (раньше) или равна указанной. При этом интервал задается и

учитывается с точностью до секунды. То есть дата открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда равна или меньше указанной.

При запросе данных в заданном диапазоне дат возвращаются только данные, попадающие в запрашиваемый интервал, при этом интервал задается и учитывается с точностью до секунды. То есть, время открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда находится в запрошенном интервале.

Таким образом, если текущий день недели Суббота, то при попытке скопировать данные на недельном таймфрейме с указанием `start_time=Последний_Вторник` и `stop_time=Последняя_Пятница` функция вернет 0, так как время открытия на недельном таймфрейме всегда приходится на воскресенье, но ни один недельный бар не попадает в указанный диапазон.

Если необходимо получить значение, соответствующее текущему незавершенному бару, то можно использовать первую форму вызова с указанием `start_pos=0` и `count=1`.

#### Пример:

```
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Пример вывода значений High[i] и Low[i]"
#property description "для баров, выбранных случайным образом"

double High[],Low[];
//+-----+
// | Получим Low для заданного номера бара |
//+-----+
double iLow(string symbol,ENUM_TIMEFRAMES timeframe,int index)
{
    double low=0;
    ArraySetAsSeries(Low,true);
    int copied=CopyLow(symbol,timeframe,0,Bars(symbol,timeframe),Low);
    if(copied>0 && index<copied) low=Low[index];
    return(low);
}
//+-----+
// | Получим High для заданного номера бара |
//+-----+
double iHigh(string symbol,ENUM_TIMEFRAMES timeframe,int index)
{
    double high=0;
    ArraySetAsSeries(High,true);
    int copied=CopyHigh(symbol,timeframe,0,Bars(symbol,timeframe),High);
    if(copied>0 && index<copied) high=High[index];
    return(high);
}
//+-----+
```

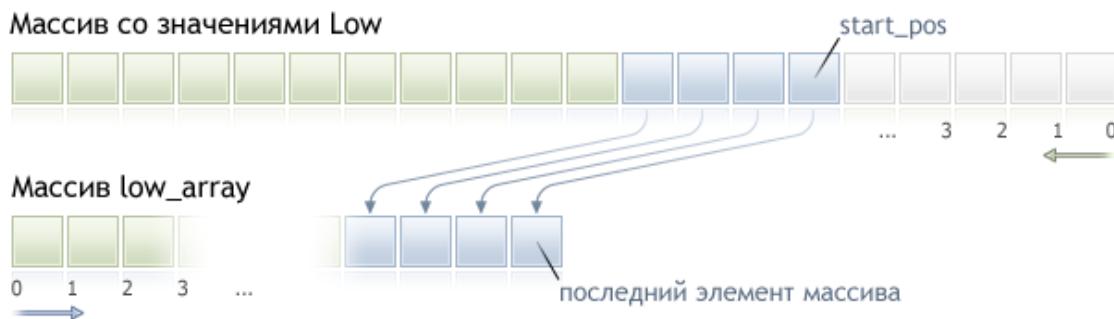
```
//| Expert tick function
//+-----+
void OnTick()
{
//--- выводим на каждом тике значения High и Low для бара с индексом,
//--- равным секунде поступления тика
datetime t=TimeCurrent();
int sec=t%60;
printf("High[%d] = %G  Low[%d] = %G",
sec,iHigh(Symbol(),0,sec),
sec,iLow(Symbol(),0,sec));
}
```

Более полный пример запроса исторических данных смотрите в разделе [Способы привязки объектов](#). В приведенном там скрипте показано, как получать значения индикатора [iFractals](#) на последних 1000 барах и как потом вывести на график по десять последних фракталов вверх и вниз. Подобный прием можно использовать для всех индикаторов, которые имеют пропуски значений и обычно реализуются с помощью следующих [стилей построения](#):

- [DRAW\\_SECTION](#),
- [DRAW\\_ARROW](#),
- [DRAW\\_ZIGZAG](#),
- [DRAW\\_COLOR\\_SECTION](#),
- [DRAW\\_COLOR\\_ARROW](#),
- [DRAW\\_COLOR\\_ZIGZAG](#).

## CopyLow

Функция получает в массив `low_array` исторические данные минимальных цен баров для указанной пары символ-период в указанном количестве. Необходимо отметить, что отсчет элементов от стартовой позиции ведется от настоящего к прошлому, то есть стартовая позиция, равная 0, означает текущий бар.



При копировании заранее неизвестного количества данных рекомендуется в качестве приемного массива использовать [динамический массив](#), так как если данных оказывается меньше (или больше), чем вмещает массив, то производится попытка перераспределения массива таким образом, чтобы запрошенные данные поместились целиком и полностью.

Если необходимо копировать заранее известное количество данных, то лучше это делать в [статически выделенный буфер](#), чтобы избежать излишнего перевыделения памяти.

Неважно, какое свойство имеет приемный массив - `as_series=true` или `as_series=false`, данные будут скопированы таким образом, что самый старый по времени элемент будет в начале физической памяти, отведенной под массив. Существует 3 варианта функции.

### Обращение по начальной позиции и количеству требуемых элементов

```
int CopyLow(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    int             start_pos,        // откуда начнем
    int             count,            // сколько копируем
    double          low_array[]       // массив для копирования минимальных цен
);
```

### Обращение по начальной дате и количеству требуемых элементов

```
int CopyLow(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,        // с какой даты
    int             count,            // сколько копируем
    double          low_array[]       // массив для копирования минимальных цен
);
```

### Обращение по начальной и конечной датам требуемого интервала времени

```
int CopyLow(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,       // с какой даты
    datetime        stop_time,        // по какую дату
    double          low_array[]      // массив для копирования минимальных цен
);
```

### Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*start\_pos*

[in] Номер первого копируемого элемента.

*count*

[in] Количество копируемых элементов.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

*low\_array[]*

[out] Массив типа [double](#).

### Возвращаемое значение

Количество скопированных элементов массива либо -1 в случае [ошибки](#).

### Примечание

Если интервал запрашиваемых данных полностью находится вне доступных данных на сервере, то функция возвращает -1. В случае если запрашиваются данные за пределами [TERMINAL\\_MAXBARS](#) (максимальное количество баров на графике), функция также вернет -1.

При запросе данных из индикатора, если запрашиваемые таймсерии еще не построены или их необходимо загрузить с сервера, функция сразу же вернет -1, но при этом сам процесс загрузки/построения будет инициирован.

При запросе данных из эксперта или скрипта, будет инициирована [загрузка с сервера](#), если локально этих данных у терминала нет, либо начнется построение нужной таймсерии, если данные можно построить из локальной истории, но они еще не готовы. Функция вернет то количество данных, которые будут готовы к моменту истечения таймаута, но загрузка истории будет продолжаться, и при следующем аналогичном запросе функция вернет уже больше данных.

При запросе данных по начальной дате и количеству требуемых элементов возвращаются только данные, дата которых меньше (раньше) или равна указанной. При этом интервал задается и

учитывается с точностью до секунды. То есть дата открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда равна или меньше указанной.

При запросе данных в заданном диапазоне дат возвращаются только данные, попадающие в запрашиваемый интервал, при этом интервал задается и учитывается с точностью до секунды. То есть время открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда находится в запрошенном интервале.

Таким образом, если текущий день недели Суббота, то при попытке скопировать данные на недельном таймфрейме с указанием `start_time=Последний_Вторник` и `stop_time=Последняя_Пятница` функция вернет 0, так как время открытия на недельном таймфрейме всегда приходится на воскресенье, но ни один недельный бар не попадает в указанный диапазон.

Если необходимо получить значение, соответствующее текущему незавершенному бару, то можно использовать первую форму вызова с указанием `start_pos=0` и `count=1`.

Пример запроса исторических данных смотрите в разделе [Способы привязки объектов](#). В приведенном там скрипте показано, как получать значения индикатора `iFractals` на последних 1000 барах и как потом вывести на график по десять последних фракталов вверх и вниз. Подобный прием можно использовать для всех индикаторов, которые имеют пропуски значений и обычно реализуются с помощью следующих [стилей построения](#):

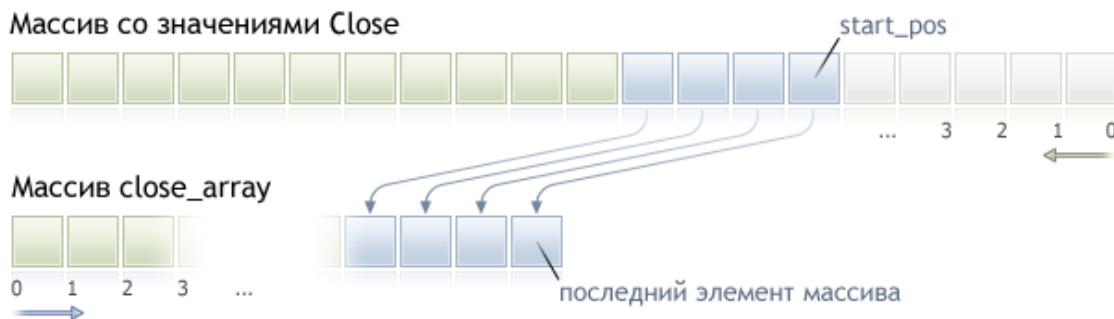
- [DRAW\\_SECTION](#),
- [DRAW\\_ARROW](#),
- [DRAW\\_ZIGZAG](#),
- [DRAW\\_COLOR\\_SECTION](#),
- [DRAW\\_COLOR\\_ARROW](#),
- [DRAW\\_COLOR\\_ZIGZAG](#).

Смотри также

[CopyHigh](#)

## CopyClose

Функция получает в массив `close_array` исторические данные цен закрытия баров для указанной пары символ-период в указанном количестве. Необходимо отметить, что отсчет элементов от стартовой позиции ведется от настоящего к прошлому, то есть стартовая позиция, равная 0, означает текущий бар.



При копировании заранее неизвестного количества данных рекомендуется в качестве приемного массива использовать [динамический массив](#), так как если данных оказывается меньше (или больше), чем вмещает массив, то производится попытка перераспределения массива таким образом, чтобы запрошенные данные поместились целиком и полностью.

Если необходимо копировать заранее известное количество данных, то лучше это делать в [статически выделенный буфер](#), чтобы избежать излишнего перевыделения памяти.

Неважно, какое свойство имеет приемный массив - `as_series=true` или `as_series=false`, данные будут скопированы таким образом, что самый старый по времени элемент будет в начале физической памяти, отведенной под массив. Существует 3 варианта функции.

### Обращение по начальной позиции и количеству требуемых элементов

```
int CopyClose(
    string          symbol_name,           // имя символа
    ENUM_TIMEFRAMES timeframe,           // период
    int             start_pos,            // откуда начнем
    int             count,                // сколько копируем
    double          close_array[]        // массив для копирования цен закрытия
);
```

### Обращение по начальной дате и количеству требуемых элементов

```
int CopyClose(
    string          symbol_name,           // имя символа
    ENUM_TIMEFRAMES timeframe,           // период
    datetime        start_time,           // с какой даты
    int             count,                // сколько копируем
    double          close_array[]        // массив для копирования цен закрытия
);
```

### Обращение по начальной и конечной датам требуемого интервала времени

```

int CopyClose(
    string          symbol_name,           // имя символа
    ENUM_TIMEFRAMES timeframe,           // период
    datetime        start_time,           // с какой даты
    datetime        stop_time,            // по какую дату
    double          close_array[]        // массив для копирования цен закрытия
);

```

### Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*start\_pos*

[in] Номер первого копируемого элемента.

*count*

[in] Количество копируемых элементов.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

*close\_array[]*

[out] Массив типа [double](#).

### Возвращаемое значение

Количество скопированных элементов массива либо -1 в случае [ошибки](#).

### Примечание

Если интервал запрашиваемых данных полностью находится вне доступных данных на сервере, то функция возвращает -1. В случае если запрашиваются данные за пределами [TERMINAL\\_MAXBARS](#) (максимальное количество баров на графике), функция также вернет -1.

При запросе данных из индикатора, если запрашиваемые таймсерии еще не построены или их необходимо загрузить с сервера, функция сразу же вернет -1, но при этом сам процесс загрузки/построения будет инициирован.

При запросе данных из эксперта или скрипта, будет инициирована [загрузка с сервера](#), если локально этих данных у терминала нет, либо начнется построение нужной таймсерии, если данные можно построить из локальной истории, но они еще не готовы. Функция вернет то количество данных, которые будут готовы к моменту истечения таймаута, но загрузка истории будет продолжаться, и при следующем аналогичном запросе функция вернет уже больше данных.

При запросе данных по начальной дате и количеству требуемых элементов возвращаются только данные, дата которых меньше (раньше) или равна указанной. При этом интервал задается и

учитывается с точностью до секунды. То есть дата открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда равна или меньше указанной.

При запросе данных в заданном диапазоне дат возвращаются только данные, попадающие в запрашиваемый интервал, при этом интервал задается и учитывается с точностью до секунды. То есть время открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда находится в запрошенном интервале.

Таким образом, если текущий день недели Суббота, то при попытке скопировать данные на недельном таймфрейме с указанием `start_time=Последний_Вторник` и `stop_time=Последняя_Пятница` функция вернет 0, так как время открытия на недельном таймфрейме всегда приходится на воскресенье, но ни один недельный бар не попадает в указанный диапазон.

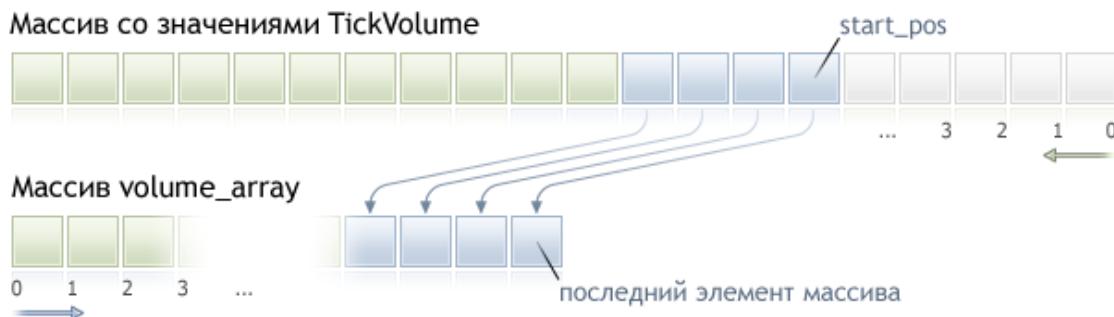
Если необходимо получить значение, соответствующее текущему незавершенному бару, то можно использовать первую форму вызова с указанием `start_pos=0` и `count=1`.

Пример запроса исторических данных смотрите в разделе [Способы привязки объектов](#). В приведенном там скрипте показано, как получать значения индикатора `iFractals` на последних 1000 барах и как потом вывести на график по десять последних фракталов вверх и вниз. Подобный прием можно использовать для всех индикаторов, которые имеют пропуски значений и обычно реализуются с помощью следующих [стилей построения](#):

- [DRAW\\_SECTION](#),
- [DRAW\\_ARROW](#),
- [DRAW\\_ZIGZAG](#),
- [DRAW\\_COLOR\\_SECTION](#),
- [DRAW\\_COLOR\\_ARROW](#),
- [DRAW\\_COLOR\\_ZIGZAG](#).

## CopyTickVolume

Функция получает в массив volume\_array исторические данные тиковых объемов для указанной пары символ-период в указанном количестве. Необходимо отметить, что отсчет элементов от стартовой позиции ведется от настоящего к прошлому, то есть стартовая позиция, равная 0, означает текущий бар.



При копировании заранее неизвестного количества данных рекомендуется в качестве приемного массива использовать [динамический массив](#), так как если данных оказывается меньше (или больше), чем вмещает массив, то производится попытка перераспределения массива таким образом, чтобы запрошенные данные поместились целиком и полностью.

Если необходимо копировать заранее известное количество данных, то лучше это делать в [статически выделенный буфер](#), чтобы избежать излишнего перевыделения памяти.

Неважно, какое свойство имеет приемный массив - as\_series=true или as\_series=false, данные будут скопированы таким образом, что самый старый по времени элемент будет в начале физической памяти, отведенной под массив. Существует 3 варианта функции.

### Обращение по начальной позиции и количеству требуемых элементов

```
int CopyTickVolume(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    int             start_pos,        // откуда начнем
    int             count,            // сколько копируем
    long            volume_array[]   // массив для копирования тиковых объемов
);
```

### Обращение по начальной дате и количеству требуемых элементов

```
int CopyTickVolume(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,       // с какой даты
    int             count,            // сколько копируем
    long            volume_array[]   // массив для копирования тиковых объемов
);
```

### Обращение по начальной и конечной датам требуемого интервала времени

```

int CopyTickVolume(
    string          symbol_name,           // имя символа
    ENUM_TIMEFRAMES timeframe,           // период
    datetime        start_time,           // с какой даты
    datetime        stop_time,            // по какую дату
    long            volume_array[]       // массив для копирования тиковых объемов
);

```

## Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*start\_pos*

[in] Номер первого копируемого элемента.

*count*

[in] Количество копируемых элементов.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

*volume\_array[]*

[out] Массив типа [long](#).

## Возвращаемое значение

Количество скопированных элементов массива либо -1 в случае [ошибки](#).

## Примечание

Если интервал запрашиваемых данных полностью находится вне доступных данных на сервере, то функция возвращает -1. В случае если запрашиваются данные за пределами [TERMINAL\\_MAXBARS](#) (максимальное количество баров на графике), функция также вернет -1.

При запросе данных из индикатора, если запрашиваемые таймсерии еще не построены или их необходимо загрузить с сервера, функция сразу же вернет -1, но при этом сам процесс загрузки/построения будет инициирован.

При запросе данных из эксперта или скрипта, будет инициирована [загрузка с сервера](#), если локально этих данных у терминала нет, либо начнется построение нужной таймсерии, если данные можно построить из локальной истории, но они еще не готовы. Функция вернет то количество данных, которые будут готовы к моменту истечения таймаута, но загрузка истории будет продолжаться, и при следующем аналогичном запросе функция вернет уже больше данных.

При запросе данных по начальной дате и количеству требуемых элементов возвращаются только данные, дата которых меньше (раньше) или равна указанной. При этом интервал задается и

учитывается с точностью до секунды. То есть дата открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда равна или меньше указанной.

При запросе данных в заданном диапазоне дат возвращаются только данные, попадающие в запрашиваемый интервал, при этом интервал задается и учитывается с точностью до секунды. То есть время открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда находится в запрошенном интервале.

Таким образом, если текущий день недели Суббота, то при попытке скопировать данные на недельном таймфрейме с указанием `start_time=Последний_Вторник` и `stop_time=Последняя_Пятница` функция вернет 0, так как время открытия на недельном таймфрейме всегда приходится на воскресенье, но ни один недельный бар не попадает в указанный диапазон.

Если необходимо получить значение, соответствующее текущему незавершенному бару, то можно использовать первую форму вызова с указанием `start_pos=0` и `count=1`.

### Пример:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- plot TickVolume
#property indicator_label1 "TickVolume"
#property indicator_type1 DRAW_HISTOGRAM
#property indicator_color1 C'143,188,139'
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameters
input int bars=3000;
//--- indicator buffers
double TickVolumeBuffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,TickVolumeBuffer,INDICATOR_DATA);
    IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
```

```

        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
//---

    if(prev_calculated==0)
    {
        long timeseries[];
        ArraySetAsSeries(timeseries,true);
        int prices=CopyTickVolume(Symbol(),0,0,bars,timeseries);
        for(int i=0;i<rates_total-prices;i++) TickVolumeBuffer[i]=0.0;
        for(int i=0;i<prices;i++) TickVolumeBuffer[rates_total-1-i]=timeseries[prices-1];
        Print("Получено исторических значений TickVolume: "+prices);
    }
    else
    {
        long timeseries[];
        int prices=CopyTickVolume(Symbol(),0,0,1,timeseries);
        TickVolumeBuffer[rates_total-1]=timeseries[0];
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}

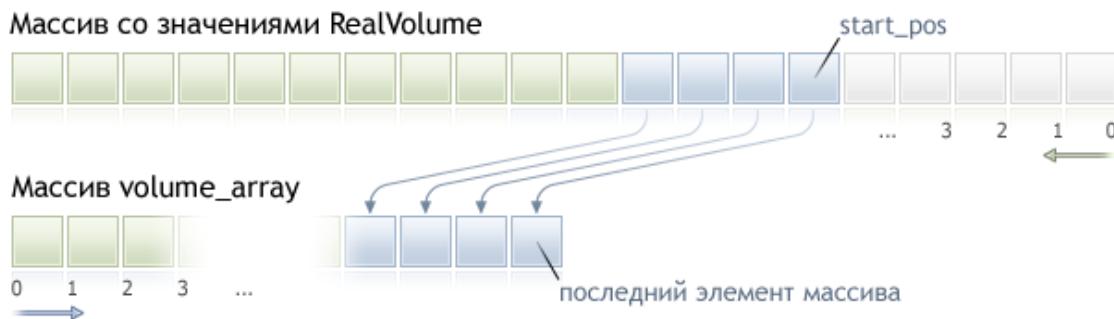
```

Более полный пример запроса исторических данных смотрите в разделе [Способы привязки объектов](#). В приведенном там скрипте показано, как получать значения индикатора [iFractals](#) на последних 1000 барах и как потом вывести на график по десять последних фракталов вверх и вниз. Подобный прием можно использовать для всех индикаторов, которые имеют пропуски значений и обычно реализуются с помощью следующих [стилей построения](#):

- [DRAW\\_SECTION](#),
- [DRAW\\_ARROW](#),
- [DRAW\\_ZIGZAG](#),
- [DRAW\\_COLOR\\_SECTION](#),
- [DRAW\\_COLOR\\_ARROW](#),
- [DRAW\\_COLOR\\_ZIGZAG](#).

## CopyRealVolume

Функция получает в массив volume\_array исторические данные торговых объемов для указанной пары символ-период в указанном количестве. Необходимо отметить, что отсчет элементов от стартовой позиции ведется от настоящего к прошлому, то есть стартовая позиция, равная 0, означает текущий бар.



При копировании заранее неизвестного количества данных рекомендуется в качестве приемного массива использовать [динамический массив](#), так как если данных оказывается меньше (или больше), чем вмещает массив, то производится попытка перераспределения массива таким образом, чтобы запрошенные данные поместились целиком и полностью.

Если необходимо копировать заранее известное количество данных, то лучше это делать в [статически выделенный буфер](#), чтобы избежать излишнего перевыделения памяти.

Неважно, какое свойство имеет приемный массив - as\_series=true или as\_series=false, данные будут скопированы таким образом, что самый старый по времени элемент будет в начале физической памяти, отведенной под массив. Существует 3 варианта функции.

### Обращение по начальной позиции и количеству требуемых элементов

```
int CopyRealVolume(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    int             start_pos,        // откуда начнем
    int             count,            // сколько копируем
    long            volume_array[]   // массив для копирования объемов
);
```

### Обращение по начальной дате и количеству требуемых элементов

```
int CopyRealVolume(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,        // с какой даты
    int             count,            // сколько копируем
    long            volume_array[]   // массив для копирования объемов
);
```

### Обращение по начальной и конечной датам требуемого интервала времени

```

int CopyRealVolume(
    string          symbol_name,           // имя символа
    ENUM_TIMEFRAMES timeframe,           // период
    datetime        start_time,           // с какой даты
    datetime        stop_time,            // по какую дату
    long            volume_array[]       // массив для копирования объемов
);

```

### Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*start\_pos*

[in] Номер первого копируемого элемента.

*count*

[in] Количество копируемых элементов.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

*volume\_array[]*

[out] Массив типа [long](#).

### Возвращаемое значение

Количество скопированных элементов массива либо -1 в случае [ошибки](#).

### Примечание

Если интервал запрашиваемых данных полностью находится вне доступных данных на сервере, то функция возвращает -1. В случае если запрашиваются данные за пределами [TERMINAL\\_MAXBARS](#) (максимальное количество баров на графике), функция также вернет -1.

При запросе данных из индикатора, если запрашиваемые таймсерии еще не построены или их необходимо загрузить с сервера, функция сразу же вернет -1, но при этом сам процесс загрузки/построения будет инициирован.

При запросе данных из эксперта или скрипта, будет инициирована [загрузка с сервера](#), если локально этих данных у терминала нет, либо начнется построение нужной таймсерии, если данные можно построить из локальной истории, но они еще не готовы. Функция вернет то количество данных, которые будут готовы к моменту истечения таймаута, но загрузка истории будет продолжаться, и при следующем аналогичном запросе функция вернет уже больше данных.

При запросе данных по начальной дате и количеству требуемых элементов возвращаются только данные, дата которых меньше (раньше) или равна указанной. При этом интервал задается и

учитывается с точностью до секунды. То есть дата открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда равна или меньше указанной.

При запросе данных в заданном диапазоне дат возвращаются только данные, попадающие в запрашиваемый интервал, при этом интервал задается и учитывается с точностью до секунды. То есть время открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда находится в запрошенном интервале.

Таким образом, если текущий день недели Суббота, то при попытке скопировать данные на недельном таймфрейме с указанием `start_time=Последний_Вторник` и `stop_time=Последняя_Пятница` функция вернет 0, так как время открытия на недельном таймфрейме всегда приходится на воскресенье, но ни один недельный бар не попадает в указанный диапазон.

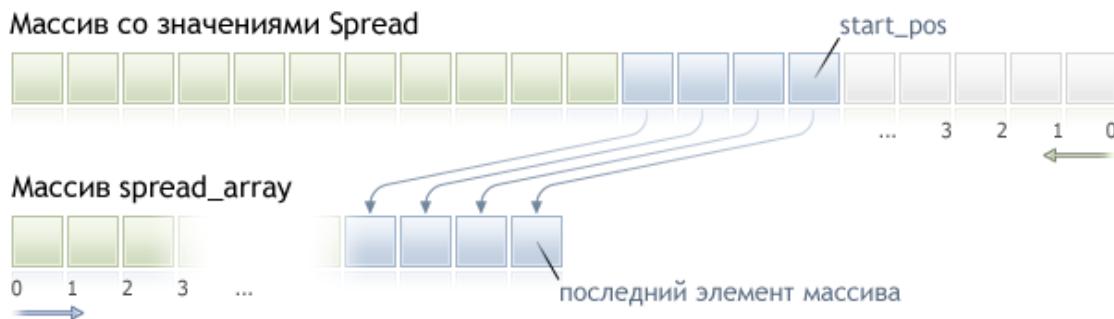
Если необходимо получить значение, соответствующее текущему незавершенному бару, то можно использовать первую форму вызова с указанием `start_pos=0` и `count=1`.

Более полный пример запроса исторических данных смотрите в разделе [Способы привязки объектов](#). В приведенном там скрипте показано, как получать значения индикатора `iFractals` на последних 1000 барах и как потом вывести на график по десять последних фракталов вверх и вниз. Подобный прием можно использовать для всех индикаторов, которые имеют пропуски значений и обычно реализуются с помощью следующих [стилей построения](#):

- [DRAW\\_SECTION](#),
- [DRAW\\_ARROW](#),
- [DRAW\\_ZIGZAG](#),
- [DRAW\\_COLOR\\_SECTION](#),
- [DRAW\\_COLOR\\_ARROW](#),
- [DRAW\\_COLOR\\_ZIGZAG](#).

## CopySpread

Функция получает в массив spread\_array исторические данные спредов для указанной пары символ-период в указанном количестве. Необходимо отметить, что отсчет элементов от стартовой позиции ведется от настоящего к прошлому, то есть стартовая позиция, равная 0, означает текущий бар.



При копировании заранее неизвестного количества данных рекомендуется в качестве приемного массива использовать [динамический массив](#), так как если данных оказывается меньше (или больше), чем вмещает массив, то производится попытка перераспределения массива таким образом, чтобы запрошенные данные поместились целиком и полностью.

Если необходимо копировать заранее известное количество данных, то лучше это делать в [статически выделенный буфер](#), чтобы избежать излишнего перевыделения памяти.

Неважно, какое свойство имеет приемный массив - as\_series=true или as\_series=false, данные будут скопированы таким образом, что самый старый по времени элемент будет в начале физической памяти, отведенной под массив. Существует 3 варианта функции.

### Обращение по начальной позиции и количеству требуемых элементов

```
int CopySpread(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    int             start_pos,        // откуда начнем
    int             count,            // сколько копируем
    int             spread_array[]   // массив для копирования спредов
);
```

### Обращение по начальной дате и количеству требуемых элементов

```
int CopySpread(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,       // с какой даты
    int             count,            // сколько копируем
    int             spread_array[]   // массив для копирования спредов
);
```

### Обращение по начальной и конечной датам требуемого интервала времени

```

int CopySpread(
    string          symbol_name,      // имя символа
    ENUM_TIMEFRAMES timeframe,       // период
    datetime        start_time,       // с какой даты
    datetime        stop_time,        // по какую дату
    int             spread_array[]   // массив для копирования спредов
);

```

### Параметры

*symbol\_name*

[in] Символ.

*timeframe*

[in] Период.

*start\_pos*

[in] Номер первого копируемого элемента.

*count*

[in] Количество копируемых элементов.

*start\_time*

[in] Время бара, соответствующее первому элементу.

*stop\_time*

[in] Время бара, соответствующее последнему элементу.

*spread\_array[]*

[out] Массив типа [int](#).

### Возвращаемое значение

Количество скопированных элементов массива либо -1 в случае [ошибки](#).

### Примечание

Если интервал запрашиваемых данных полностью находится вне доступных данных на сервере, то функция возвращает -1. В случае если запрашиваются данные за пределами [TERMINAL\\_MAXBARS](#) (максимальное количество баров на графике), функция также вернет -1.

При запросе данных из индикатора, если запрашиваемые таймсерии еще не построены или их необходимо загрузить с сервера, функция сразу же вернет -1, но при этом сам процесс загрузки/построения будет инициирован.

При запросе данных из эксперта или скрипта, будет инициирована [загрузка с сервера](#), если локально этих данных у терминала нет, либо начнется построение нужной таймсерии, если данные можно построить из локальной истории, но они еще не готовы. Функция вернет то количество данных, которые будут готовы к моменту истечения таймаута, но загрузка истории будет продолжаться, и при следующем аналогичном запросе функция вернет уже больше данных.

При запросе данных по начальной дате и количеству требуемых элементов возвращаются только данные, дата которых меньше (раньше) или равна указанной. При этом интервал задается и

учитывается с точностью до секунды. То есть дата открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда равна или меньше указанной.

При запросе данных в заданном диапазоне дат возвращаются только данные, попадающие в запрашиваемый интервал, при этом интервал задается и учитывается с точностью до секунды. То есть время открытия любого бара, для которого возвращается значение (объем, спред, значение в индикаторном буфере, цена Open, High, Low, Close или время открытия Time), всегда находится в запрошенном интервале.

Таким образом, если текущий день недели Суббота, то при попытке скопировать данные на недельном таймфрейме с указанием `start_time=Последний_Вторник` и `stop_time=Последняя_Пятница` функция вернет 0, так как время открытия на недельном таймфрейме всегда приходится на воскресенье, но ни один недельный бар не попадает в указанный диапазон.

Если необходимо получить значение, соответствующее текущему незавершенному бару, то можно использовать первую форму вызова с указанием `start_pos=0` и `count=1`.

#### Пример:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- plot Spread
#property indicator_label1 "Spread"
#property indicator_type1 DRAW_HISTOGRAM
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameters
input int bars=3000;
//--- indicator buffers
double SpreadBuffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0, SpreadBuffer, INDICATOR_DATA);
    IndicatorSetInteger(INDICATOR_DIGITS, 0);
//---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
```

```

        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
//---

    if(prev_calculated==0)
    {
        int spread_int[];
        ArraySetAsSeries(spread_int,true);
        int spreads=CopySpread(Symbol(),0,0,bars,spread_int);
        Print("Получено исторических значений спреда: ",spreads);
        for (int i=0;i<spreads;i++)
        {
            SpreadBuffer[rates_total-1-i]=spread_int[i];
            if(i<=30) Print("spread["+i+"] =",spread_int[i]);
        }
    }
    else
    {
        double Ask,Bid;
        Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
        Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
        Comment("Ask = ",Ask," Bid = ",Bid);
        SpreadBuffer[rates_total-1]=(Ask-Bid)/Point();
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}

```

Более полный пример запроса исторических данных смотрите в разделе [Способы привязки объектов](#). В приведенном там скрипте показано, как получать значения индикатора [iFractals](#) на последних 1000 барах и как потом вывести на график по десять последних фракталов вверх и вниз. Подобный прием можно использовать для всех индикаторов, которые имеют пропуски значений и обычно реализуются с помощью следующих [стилей построения](#):

- [DRAW\\_SECTION](#),
- [DRAW\\_ARROW](#),
- [DRAW\\_ZIGZAG](#),
- [DRAW\\_COLOR\\_SECTION](#),
- [DRAW\\_COLOR\\_ARROW](#),
- [DRAW\\_COLOR\\_ZIGZAG](#).

## CopyTicks

Функция получает в массив ticks\_array тики в формате [MqlTick](#), при этом индексация ведётся от прошлого к настоящему, то есть тик с индексом 0 является самым старым в массиве. Для анализа тика необходимо проверять поле *flags*, которое сообщает, что именно было изменено в данном тике.

```
int CopyTicks(
    string          symbol_name,           // имя символа
    MqlTick&        ticks_array[],        // массив для приёма тиков
    uint            flags=COPY_TICKS_ALL,   // флаг, определяющий тип получаемых тиков
    ulong           from=0,                // дата, начиная с которой запрашиваются тики
    uint            count=0               // количество тиков, которые необходимо получить
);
```

### Параметры

*symbol\_name*

[in] Символ.

*ticks\_array*

[out] Массив типа [MqlTick](#) для приема тиков.

*flags*

[in] Флаг, определяющий тип запрашиваемых тиков. **COPY\_TICKS\_INFO** - тики, вызванные изменениями Bid и/или Ask, **COPY\_TICKS\_TRADE** - тики с изменениями Last и Volume, **COPY\_TICKS\_ALL** - все тики. При любом типе запроса в оставшиеся поля структуры MqlTick дописываются значения предыдущего тика.

*from*

[in] Дата, начиная с которой запрашиваются тики. Указывается в миллисекундах с 01.01.1970. Если параметр *from=0*, то отдаются последние *count* тиков.

*count*

[in] Количество запрашиваемых тиков. Если параметры *from* и *count* не указаны, то в массив ticks\_array[] будут записаны все доступные последние тики, но не более 2000.

### Возвращаемое значение

Количество скопированных тиков либо -1 в случае [ошибки](#).

### Примечание

Функция CopyTicks() позволяет запрашивать и анализировать все пришедшие тики. Первый вызов CopyTicks() инициирует синхронизацию базы тиков, хранящихся на жёстком диске по данному символу. Если тиков в локальной базе не хватает, то недостающие тики автоматически будут загружены с торгового сервера. При этом будут синхронизированы тики с даты *from*, указанной в CopyTicks(), по текущий момент. После этого все приходящие по данному символу тики будут поступать в тиковую базу и поддерживать её в актуальном синхронизированном состоянии.

Если параметры *from* и *count* не указаны, то в массив ticks\_array[] будут записаны все доступные тики, но не более 2000. Параметр *flags* позволяет задать тип требуемых тиков.

**COPY\_TICKS\_INFO** - отдаются тики, в которых есть изменения цены Bid и/или Ask. Но при этом будут также заполнены данные остальных полей, например, если изменилась только цена Bid, то в поля *ask* и *volume* будут записаны последние известные значения. Чтобы узнать точно, что именно изменилось, необходимо анализировать поле *flags*, которое будет иметь значение **TICK\_FLAG\_BID** и/или **TICK\_FLAG\_ASK**. Если тик имеет нулевые значения цен Bid и Ask, и при этом флаги показывают, что эти данные цены изменились (*flags=TICK\_FLAG\_BID|TICK\_FLAG\_ASK*), то это говорит об опустошении стакана заявок. Другими словами, в этот момент отсутствуют заявки на покупку и продажу.

**COPY\_TICKS\_TRADE** - отдаются тики, в которых есть изменения последней цены сделки и объема. Но при этом будут также заполнены данные остальных полей, то есть в поля Bid и Ask будут записаны последние известные значения. Чтобы узнать точно, что именно изменилось, необходимо анализировать поле *flags*, которое будет иметь значение **TICK\_FLAG\_LAST** и **TICK\_FLAG\_VOLUME**.

**COPY\_TICKS\_ALL** - отдаются все тики, в которых есть хоть какое-то изменение. При этом неизмененные поля также заполняются последними известными значениями.

Вызов `CopyTicks()` с флагом **COPY\_TICKS\_ALL** выдает сразу все тики из запрашиваемого диапазона, в то время как вызов в других режимах требует некоторого времени на предобработку и выборку тиков, и поэтому не даёт существенного выигрыша по скорости выполнения.

При запросе тиков (неважно, **COPY\_TICKS\_INFO** или **COPY\_TICKS\_TRADE**), в каждом тике содержится полная ценовая информация на момент тика (*bid*, *ask*, *last* и *volume*). Это сделано для удобства анализа торговой обстановки на момент каждого тика, чтобы не приходилось каждый раз запрашивать глубокую тиковую историю и искать в ней значения по другим полям.

**В индикаторах функция CopyTicks() возвращает результат немедленно:** При вызове из индикатора `CopyTick()` сразу же вернёт доступные по символу тики, а также запустит синхронизацию базы тиков, если данных не хватило. Все индикаторы на одном символе работают в одном общем потоке, поэтому индикатор не имеет права ждать завершения синхронизации. После окончания синхронизации при последующем вызове `CopyTicks()` вернёт все запрашиваемые тики. Функция [OnCalculate\(\)](#) в индикаторах вызывается после поступления каждого тика.

**В экспертах и скриптах функция CopyTicks() может дожидаться результата до 45 секунд:** В отличие от индикатора каждый эксперт и скрипт работает в собственном потоке, и поэтому может дожидаться окончания синхронизации до 45 секунд. Если за это время тики так и не будут синхронизированы в необходимом объеме, то `CopyTicks()` по таймауту вернёт только имеющиеся в наличии тики, при этом синхронизация продолжится. Функция [OnTick\(\)](#) в экспертах не является обработчиком каждого тика, она лишь уведомляет эксперта об изменениях на рынке. Изменения могут быть пакетными: в терминал может одновременно прийти несколько тиков, но функция `OnTick()` будет вызвана лишь один раз для уведомления эксперта о последнем состоянии рынка.

**Скорость выдачи:** терминал хранит по каждому символу 4096 последних тиков в кеше для быстрого доступа (для символов с запущенным стаканом - 65536 тиков), запросы к этим данным выполняются быстрее всего. При запросе тиков текущей торговой сессии за пределами кеша `CopyTicks()` обращается уже к тикам, которые хранятся в памяти терминала, эти запросы требуют большего времени на выполнение. Самыми медленными являются запросы тиков за другие дни, так как в этом случае данные читаются уже с диска.

**Пример:**

```

#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs

//--- запрашиваем 100 миллионов тиков, чтобы гарантировать получение всей тиковой истории
input int      getticks=100000000; // сколько тиков требуется

//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//---

    int      attempts=0;      // счетчик попыток
    bool     success=false;   // флаг успешного выполнения копирования тиков
    MqlTick tick_array[];    // массив для приема тиков
    MqlTick lasttick;        // для получения данных последнего тика
    SymbolInfoTick(_Symbol,lasttick);

//--- сделаем 3 попытки получить тики
    while(attempts<3)
    {
        //--- замерим время старта перед получением тиков
        uint start=GetTickCount();

//--- запросим тиковую историю с момента 1970.01.01 00:00:001 (параметр from=1 ms)
        int received=CopyTicks(_Symbol,tick_array,COPY_TICKS_ALL,1,getticks);
        if(received!=-1)
        {
            //--- выведем информацию о количестве тиков и затраченном времени времени
            PrintFormat("%s: received %d ticks in %d ms",_Symbol,received,GetTickCount());
            //--- если тиковая история синхронизирована, то код ошибки равен нулю
            if(GetLastError()==0)
            {
                success=true;
                break;
            }
        }
        else
            PrintFormat("%s: Ticks are not synchronized yet, %d ticks received for %d ms",_Symbol,received,GetTickCount()-start,_LastError);
    }

//--- считаем попытки
attempts++;

//--- пауза в 1 секунду в ожидании завершения синхронизации тиковой базы
Sleep(1000);
}

//--- не удалось получить запрошенные тики от самого начала истории с трех попыток
if(!success)
{
    PrintFormat("Ошибка! Не удалось получить %d тиков по %s с трех попыток",getticks,_LastError);
    return;
}

```

```

int ticks=ArraySize(tick_array);
//--- выводем время первого тика в массиве
datetime firstticktime=tick_array[ticks-1].time;
PrintFormat("Last tick time = %s.%03I64u",
           TimeToString(firstticktime,TIME_DATE|TIME_MINUTES|TIME_SECONDS),tick_array);
//--- выводем время последнего тика в массиве
datetime lastticktime=tick_array[0].time;
PrintFormat("First tick time = %s.%03I64u",
           TimeToString(lastticktime,TIME_DATE|TIME_MINUTES|TIME_SECONDS),tick_array);

//---
MqlDateTime today;
datetime current_time=TimeCurrent();
TimeToStruct(current_time,today);
PrintFormat("current_time=%s",TimeToString(current_time));
today.hour=0;
today.min=0;
today.sec=0;
datetime startday=StructToTime(today);
datetime endday=startday+24*60*60;
if((ticks=CopyTicksRange(_Symbol,tick_array,COPY_TICKS_ALL,startday*1000,endday*1000))<0)
{
    PrintFormat("CopyTicksRange(%s,tick_array,COPY_TICKS_ALL,%s,%s) failed, error %d",
               _Symbol,TimeToString(startday),TimeToString(endday),GetLastError());
    return;
}
ticks=MathMax(100,ticks);
//--- теперь выведем первые 100 тиков последнего дня
int counter=0;
for(int i=0;i<ticks;i++)
{
    datetime time=tick_array[i].time;
    if((time>=startday) && (time<endday) && counter<100)
    {
        counter++;
        PrintFormat("%d. %s",counter,GetTickDescription(tick_array[i]));
    }
}
//--- выведем первые 100 сделок последнего дня
counter=0;
for(int i=0;i<ticks;i++)
{
    datetime time=tick_array[i].time;
    if((time>=startday) && (time<endday) && counter<100)
    {
        if(((tick_array[i].flags&TICK_FLAG_BUY)==TICK_FLAG_BUY) || ((tick_array[i].flags&TICK_FLAG_SELL)==TICK_FLAG_SELL))
        {
            counter++;
            PrintFormat("%d. %s",counter,GetTickDescription(tick_array[i]));
        }
    }
}

```

```

        }
    }
}

//+-----+
//| возвращает строковое описание тика |
//+-----+

string GetTickDescription(MqlTick &tick)
{
    string desc=StringFormat("%s.%03d ",
                             TimeToString(tick.time), tick.time_msc%1000);
    //--- проверим флаги
    bool buy_tick=((tick.flags&TICK_FLAG_BUY)==TICK_FLAG_BUY);
    bool sell_tick=((tick.flags&TICK_FLAG_SELL)==TICK_FLAG_SELL);
    bool ask_tick=((tick.flags&TICK_FLAG_ASK)==TICK_FLAG_ASK);
    bool bid_tick=((tick.flags&TICK_FLAG_BID)==TICK_FLAG_BID);
    bool last_tick=((tick.flags&TICK_FLAG_LAST)==TICK_FLAG_LAST);
    bool volume_tick=((tick.flags&TICK_FLAG_VOLUME)==TICK_FLAG_VOLUME);
    //--- проверим сначала тик на торговые флаги
    if(buy_tick || sell_tick)
    {
        //--- сформируем вывод для торгового тика
        desc=desc+(buy_tick?StringFormat("Buy Tick: Last=%G Volume=%d ",tick.last,tick.volume));
        desc=desc+(sell_tick?StringFormat("Sell Tick: Last=%G Volume=%d ",tick.last,tick.volume));
        desc=desc+(ask_tick?StringFormat("Ask=%G ",tick.ask):"");
        desc=desc+(bid_tick?StringFormat("Bid=%G ",tick.ask):"");
        desc=desc+"(Trade tick)";
    }
    else
    {
        //--- для инфо тика сформируем вывод немного иначе
        desc=desc+(ask_tick?StringFormat("Ask=%G ",tick.ask):"");
        desc=desc+(bid_tick?StringFormat("Bid=%G ",tick.ask):"");
        desc=desc+(last_tick?StringFormat("Last=%G ",tick.last):"");
        desc=desc+(volume_tick?StringFormat("Volume=%d ",tick.volume):"");
        desc=desc+"(Info tick)";
    }
    //--- вернем описание тика
    return desc;
}
//+-----+
/* Пример вывода
Si-12.16: received 11048387 ticks in 4937 ms
Last tick time = 2016.09.26 18:32:59.775
First tick time = 2015.06.18 09:45:01.000
1. 2016.09.26 09:45.249 Ask=65370 Bid=65370 (Info tick)
2. 2016.09.26 09:47.420 Ask=65370 Bid=65370 (Info tick)
3. 2016.09.26 09:50.893 Ask=65370 Bid=65370 (Info tick)
4. 2016.09.26 09:51.827 Ask=65370 Bid=65370 (Info tick)
*/

```

```
5. 2016.09.26 09:53.810 Ask=65370 Bid=65370 (Info tick)
6. 2016.09.26 09:54.491 Ask=65370 Bid=65370 (Info tick)
7. 2016.09.26 09:55.913 Ask=65370 Bid=65370 (Info tick)
8. 2016.09.26 09:59.350 Ask=65370 Bid=65370 (Info tick)
9. 2016.09.26 09:59.678 Bid=65370 (Info tick)
10. 2016.09.26 10:00.000 Sell Tick: Last=65367 Volume=3 (Trade tick)
11. 2016.09.26 10:00.000 Sell Tick: Last=65335 Volume=45 (Trade tick)
12. 2016.09.26 10:00.000 Sell Tick: Last=65334 Volume=95 (Trade tick)
13. 2016.09.26 10:00.191 Sell Tick: Last=65319 Volume=1 (Trade tick)
14. 2016.09.26 10:00.191 Sell Tick: Last=65317 Volume=1 (Trade tick)
15. 2016.09.26 10:00.191 Sell Tick: Last=65316 Volume=1 (Trade tick)
16. 2016.09.26 10:00.191 Sell Tick: Last=65316 Volume=10 (Trade tick)
17. 2016.09.26 10:00.191 Sell Tick: Last=65315 Volume=5 (Trade tick)
18. 2016.09.26 10:00.191 Sell Tick: Last=65313 Volume=3 (Trade tick)
19. 2016.09.26 10:00.191 Sell Tick: Last=65307 Volume=25 (Trade tick)
20. 2016.09.26 10:00.191 Sell Tick: Last=65304 Volume=1 (Trade tick)
21. 2016.09.26 10:00.191 Sell Tick: Last=65301 Volume=1 (Trade tick)
22. 2016.09.26 10:00.191 Sell Tick: Last=65301 Volume=10 (Trade tick)
23. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=5 (Trade tick)
24. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=1 (Trade tick)
25. 2016.09.26 10:00.191 Sell Tick: Last=65300 Volume=6 (Trade tick)
26. 2016.09.26 10:00.191 Sell Tick: Last=65299 Volume=1 (Trade tick)
27. 2016.09.26 10:00.191 Bid=65370 (Info tick)
28. 2016.09.26 10:00.232 Ask=65297 (Info tick)
29. 2016.09.26 10:00.276 Sell Tick: Last=65291 Volume=31 (Trade tick)
30. 2016.09.26 10:00.276 Sell Tick: Last=65290 Volume=1 (Trade tick)
*/
```

#### Смотри также

[SymbolInfoTick](#), [Структура для получения текущих цен](#), [OnTick\(\)](#)

## CopyTicksRange

Функция получает в массив ticks\_array тики в формате [MqlTick](#) в указанном диапазоне дат. При этом индексация ведётся от прошлого к настоящему, то есть тик с индексом 0 является самым старым в массиве. Для анализа тика необходимо проверять поле *flags*, которое сообщает о том, что именно изменилось.

```
int CopyTicksRange(
    const string      symbol_name,           // имя символа
    MqlTick&          ticks_array[],        // массив для приёма тиков
    uint              flags=COPY_TICKS_ALL,   // флаг, определяющий тип получаемых тиков
    ulong             from_msc=0,            // дата, начиная с которой запрашиваются тики
    ulong             to_msc=0               // дата, по которую запрашиваются тики
);
```

### Параметры

*symbol\_name*

[in] Символ.

*ticks\_array*

[out] Статический или динамический массив [MqlTick](#) для приема тиков. Если в статический массив не вмещаются все тики из запрошенного интервала времени, то будет получено столько тиков, сколько помещается в массив. При этом функция сгенерирует ошибку [ERR\\_HISTORY\\_SMALL\\_BUFFER](#) (4407) .

*flags*

[in] Флаг, определяющий тип запрашиваемых тиков. [COPY\\_TICKS\\_INFO](#) - тики, вызванные изменениями Bid и/или Ask, [COPY\\_TICKS\\_TRADE](#) - тики с изменениями Last и Volume, [COPY\\_TICKS\\_ALL](#) - все тики. При любом типе запроса в оставшиеся поля структуры MqlTick дописываются значения предыдущего тика.

*from\_msc*

[in] Дата, начиная с которой запрашиваются тики. Указывается в миллисекундах с 01.01.1970. Если параметр *from\_msc* не указан, то отдаются тики от самого начала истории. Отдаются тики со временем  $\geq$  *from\_msc*.

*to\_msc*

[in] Дата, по которую запрашиваются тики. Указывается в миллисекундах с 01.01.1970. Отдаются тики со временем  $\leq$  *to\_msc*. Если параметр *to\_msc* не указан, то отдаются все тики до конца истории.

### Возвращаемое значение

Количество скопированных тиков либо -1 в случае ошибки. [GetLastError\(\)](#) может возвращать следующие ошибки:

- [ERR\\_HISTORY\\_TIMEOUT](#) - время ожидание синхронизации тиков вышло, функция отдала всё что было.
- [ERR\\_HISTORY\\_SMALL\\_BUFFER](#) - статический буфер слишком маленький, отдано столько, сколько поместились в массив.

- `ERR_NOT_ENOUGH_MEMORY` - не хватает памяти для получения истории из указанного диапазона в динамический массив тиков. Не удалось выделить нужный объем памяти под массиве тиков.

#### Примечание

Функция `CopyTicksRange()` предназначена для запроса тиков из строго указанного диапазона, например, за конкретный день истории. В то время как `CopyTicks()` позволяет указать только начальную дату, например - получить все тики с начала месяца по текущий момент.

#### Смотри также

[SymbolInfoTick](#), [Структура для получения текущих цен](#), [OnTick](#), [CopyTicks](#)

## iBars

Возвращает количество баров в истории по соответствующему символу и периоду.

```
int iBars(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe        // период
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

### Возвращаемое значение

Количество баров в истории по соответствующему символу и периоду, но не более чем задано в настройках платформы параметром "Макс. баров в окне" ("Max bars in chart")

### Пример:

```
Print("Bar count on the 'EURUSD,H1' is ",iBars("EURUSD",PERIOD_H1));
```

### Смотри также

[Bars](#)

## iBarShift

Поиск бара по времени. Функция возвращает индекс бара, в который попадает указанное время.

```
int iBarShift(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    datetime          time,             // время
    bool              exact=false      // режим
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). PERIOD\_CURRENT означает период текущего графика.

*time*

[in] Значение времени для поиска.

*exact=false*

[in] Возвращаемое значение, если бар на указанное время не найден. При значении *exact=false* *iBarShift* возвращает индекс ближайшего бара, у которого время открытия меньше указанного (*time\_open<time*). Если такой бар не найден (нет истории раньше указанного времени), то функция вернет -1. Если *exact=true*, то ближайший бар не ищется и функция *iBarShift* сразу возвращает -1.

### Возвращаемое значение

Индекс бара, в который попадает указанное время. Если для указанного времени бар отсутствует ("дыра" в истории), то функция возвращает -1 или индекс ближайшего бара (в зависимости от параметра *exact*).

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- дата приходится на воскресенье
datetime time=D'2002.04.25 12:00';
string symbol="GBPUSD";
ENUM_TIMEFRAMES tf=PERIOD_H1;
bool exact=false;
//--- если бара на указанное время нет - iBarShift вернет индекс ближайшего бара
int bar_index=iBarShift(symbol,tf,time,exact);
//--- проверим код ошибки после вызова iBarShift()
int error=GetLastError();
```

```

if(error!=0)
{
    PrintFormat("iBarShift(): GetLastError=%d - запрашиваемая дата %s "+
        "для %s %s в доступной истории не найдена",
        error,TimeToString(time),symbol,EnumToString(tf));
    return;
}
//--- функция iBarShift() выполнена успешно, выведем результаты для exact=false
PrintFormat("1. %s %s %s(%s): bar index is %d (exact=%s)",
    symbol,EnumToString(tf),TimeToString(time),
    DayOfWeek(time),bar_index,string(exact));
datetime bar_time=iTime(symbol,tf,bar_index);
PrintFormat("Time of bar #%d is %s (%s)",
    bar_index,TimeToString(bar_time),DayOfWeek(bar_time));
//--- требуем найти индекс бара на указанное время, если его нет, то получим -1
exact=true;
bar_index=iBarShift(symbol,tf,time,exact);
//--- функция iBarShift() выполнена успешно, выведем результаты для exact=true
PrintFormat("2. %s %s %s (%s):bar index is %d (exact=%s)",
    symbol,EnumToString(tf),TimeToString(time)
    ,DayOfWeek(time),bar_index,string(exact));
}

//+-----+
//| Возвращает название дня недели |
//+-----+
string DayOfWeek(const datetime time)
{
    MqlDateTime dt;
    string day="";
    TimeToStruct(time,dt);
    switch(dt.day_of_week)
    {
        case 0: day=EnumToString(SUNDAY);
        break;
        case 1: day=EnumToString(MONDAY);
        break;
        case 2: day=EnumToString(TUESDAY);
        break;
        case 3: day=EnumToString(WEDNESDAY);
        break;
        case 4: day=EnumToString(THURSDAY);
        break;
        case 5: day=EnumToString(FRIDAY);
        break;
        default:day=EnumToString(SATURDAY);
        break;
    }
}
//---
return day;

```

```
    }

//+-----+
/* Результат выполнения
1. GBPUSD PERIOD_H1 2018.06.10 12:00 (SUNDAY) : bar index is 64 (exact=false)
Time of bar #64 is 2018.06.08 23:00 (FRIDAY)
2. GBPUSD PERIOD_H1 2018.06.10 12:00 (SUNDAY) :bar index is -1 (exact=true)
*/
```

## iClose

Возвращает значение цены закрытия бара (указанного параметром shift) соответствующего графика.

```
double iClose(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    int               shift             // сдвиг
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

*shift*

[in] Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество баров назад).

### Возвращаемое значение

Значение цены закрытия бара (указанного параметром shift) соответствующего графика или 0 в случае ошибки. Для получения дополнительной информации об [ошибке](#) необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция всегда возвращает актуальные данные, для этого она при каждом вызове делает запрос к таймсерии по указанным символу/периоду. Это означает, что при отсутствии готовых данных на первом вызове функции может понадобиться некоторое время для подготовки результата выполнения.

Функция не хранит результатов предыдущих вызовов, локального кеша для быстрого возврата значения нет.

### Пример:

```
input int shift=0;
//+-----+
//| Function-event handler "tick" |
//+-----+
void OnTick()
{
    datetime time = iTime(Symbol(), Period(), shift);
    double open = iOpen(Symbol(), Period(), shift);
    double high = iHigh(Symbol(), Period(), shift);
    double low = iLow(Symbol(), Period(), shift);
    double close = iClose(NULL, PERIOD_CURRENT, shift);
```

```
long      volume= iVolume(Symbol(),0,shift);
int       bars   = iBars(NULL,0);

Comment(Symbol(),"","",EnumToString(Period()),"\n",
        "Time: " ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
        "Open: " ,DoubleToString(open,Digits()),"\n",
        "High: " ,DoubleToString(high,Digits()),"\n",
        "Low: " ,DoubleToString(low,Digits()),"\n",
        "Close: " ,DoubleToString(close,Digits()),"\n",
        "Volume: " ,IntegerToString(volume),"\n",
        "Bars: " ,IntegerToString(bars),"\n"
    );
}
```

Смотри также

[CopyClose](#), [CopyRates](#)

## iHigh

Возвращает значение максимальной цены бара (указанного параметром shift) соответствующего графика.

```
double iHigh(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    int               shift             // сдвиг
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

*shift*

[in] Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество баров назад).

### Возвращаемое значение

Значение максимальной цены бара (указанного параметром shift) соответствующего графика или 0 в случае ошибки. Для получения дополнительной информации об [ошибке](#) необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция всегда возвращает актуальные данные, для этого она при каждом вызове делает запрос к таймсерии по указанным символу/периоду. Это означает, что при отсутствии готовых данных на первом вызове функции может понадобиться некоторое время для подготовки результата выполнения.

Функция не хранит результатов предыдущих вызовов, локального кеша для быстрого возврата значения нет.

### Пример:

```
input int shift=0;
//+-----+
//| Function-event handler "tick" |
//+-----+
void OnTick()
{
    datetime time = iTime(Symbol(), Period(), shift);
    double open = iOpen(Symbol(), Period(), shift);
    double high = iHigh(Symbol(), Period(), shift);
    double low = iLow(Symbol(), Period(), shift);
    double close = iClose(NULL, PERIOD_CURRENT, shift);
```

```
long      volume= iVolume(Symbol(),0,shift);
int       bars   = iBars(NULL,0);

Comment(Symbol(),"","",EnumToString(Period()),"\n",
        "Time: " ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
        "Open: " ,DoubleToString(open,Digits()),"\n",
        "High: " ,DoubleToString(high,Digits()),"\n",
        "Low: " ,DoubleToString(low,Digits()),"\n",
        "Close: " ,DoubleToString(close,Digits()),"\n",
        "Volume: " ,IntegerToString(volume),"\n",
        "Bars: " ,IntegerToString(bars),"\n"
    );
}
```

Смотри также

[CopyHigh](#), [CopyRates](#)

## iHighest

Возвращает индекс наибольшего найденного значения (смещение относительно текущего бара) соответствующего графика.

```
int iHighest(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    ENUM_SERIESMODE   type,             // идентификатор таймсерии
    int               count=WHOLE_ARRAY, // число элементов
    int               start=0           // индекс
);
```

### Параметры

*symbol*

[in] Символ, на котором будет производиться поиск. NULL означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления ENUM\_TIMEFRAMES. 0 означает период текущего графика.

*type*

[in] Идентификатор таймсерии, в которой будет производится поиск. Может быть любым из значений ENUM\_SERIESMODE.

*count=WHOLE\_ARRAY*

[in] Число элементов таймсерии (в направлении от текущего бара в сторону возрастания индекса), среди которых должен быть произведен поиск.

*start=0*

[in] Индекс (смещение относительно текущего бара) начального бара, с которого начинается поиск наибольшего значения. Отрицательные значения игнорируются и заменяются нулевым значением.

### Возвращаемое значение

Индекс наибольшего найденного значения (смещение относительно текущего бара) соответствующего графика или -1 в случае ошибки. Для получения дополнительной информации об ошибке необходимо вызвать функцию GetLastError().

### Пример:

```
double val;
//--- расчет максимального значения цены Close на 20 последовательных барах
//--- начиная с индекса 4 и заканчивая по индекс 23 включительно на текущем графике
int val_index=iHighest(NULL,0,MODE_CLOSE,20,4);
if(val_index!=-1)
    val=High[val_index];
else
    PrintFormat("Ошибка вызова iHighest(). Код ошибки=%d",GetLastError());
```

## iLow

Возвращает значение минимальной цены бара (указанного параметром shift) соответствующего графика.

```
double iLow(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    int               shift             // сдвиг
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

*shift*

[in] Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество баров назад).

### Возвращаемое значение

Значение минимальной цены бара (указанного параметром shift) соответствующего графика или 0 в случае ошибки. Для получения дополнительной информации об [ошибке](#) необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция всегда возвращает актуальные данные, для этого она при каждом вызове делает запрос к таймсерии по указанным символу/периоду. Это означает, что при отсутствии готовых данных на первом вызове функции может понадобиться некоторое время для подготовки результата выполнения.

Функция не хранит результатов предыдущих вызовов, локального кеша для быстрого возврата значения нет.

### Пример:

```
input int shift=0;
//+-----+
//| Function-event handler "tick" |
//+-----+
void OnTick()
{
    datetime time = iTime(Symbol(), Period(), shift);
    double open = iOpen(Symbol(), Period(), shift);
    double high = iHigh(Symbol(), Period(), shift);
    double low = iLow(Symbol(), Period(), shift);
    double close = iClose(NULL, PERIOD_CURRENT, shift);
```

```
long      volume= iVolume(Symbol(),0,shift);
int       bars   = iBars(NULL,0);

Comment(Symbol(),"","",EnumToString(Period()),"\n",
        "Time: " ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
        "Open: " ,DoubleToString(open,Digits()),"\n",
        "High: " ,DoubleToString(high,Digits()),"\n",
        "Low: " ,DoubleToString(low,Digits()),"\n",
        "Close: " ,DoubleToString(close,Digits()),"\n",
        "Volume: " ,IntegerToString(volume),"\n",
        "Bars: " ,IntegerToString(bars),"\n"
    );
}
```

Смотри также

[CopyLow](#), [CopyRates](#)

## iLowest

Возвращает индекс наименьшего найденного значения (смещение относительно текущего бара) соответствующего графика.

```
int iLowest(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    ENUM_SERIESMODE   type,             // идентификатор таймсерии
    int               count=WHOLE_ARRAY, // число элементов
    int               start=0           // индекс
);
```

### Параметры

*symbol*

[in] Символ, на котором будет производиться поиск. NULL означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления ENUM\_TIMEFRAMES. 0 означает период текущего графика.

*type*

[in] Идентификатор таймсерии, в которой будет производится поиск. Может быть любым из значений ENUM\_SERIESMODE.

*count=WHOLE\_ARRAY*

[in] Число элементов таймсерии (в направлении от текущего бара в сторону возрастания индекса), среди которых должен быть произведен поиск.

*start=0*

[in] Индекс (смещение относительно текущего бара) начального бара, с которого начинается поиск наименьшего значения. Отрицательные значения игнорируются и заменяются нулевым значением.

### Возвращаемое значение

Индекс наименьшего найденного значения (смещение относительно текущего бара) соответствующего графика или -1 в случае ошибки. Для получения дополнительной информации об ошибке необходимо вызвать функцию GetLastError().

### Пример:

```
double val;
//--- поиск бара с минимальным значением реального объема на 15 последовательных барах
//--- начиная с индекса 10 и заканчивая по индекс 24 включительно на текущем графике
int val_index=iLowest(NULL,0,MODE_REAL_VOLUME,15,10);
if(val_index!=-1)
    val=Low[val_index];
else
    PrintFormat("Ошибка вызова iLowest(). Код ошибки=%d",GetLastError());
```

## iOpen

Возвращает значение цены открытия бара (указанного параметром shift) соответствующего графика.

```
double iOpen(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    int               shift             // сдвиг
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

*shift*

[in] Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество баров назад).

### Возвращаемое значение

Значение цены открытия бара (указанного параметром shift) соответствующего графика или 0 в случае ошибки. Для получения дополнительной информации об [ошибке](#) необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция всегда возвращает актуальные данные, для этого она при каждом вызове делает запрос к таймсерии по указанным символу/периоду. Это означает, что при отсутствии готовых данных на первом вызове функции может понадобиться некоторое время для подготовки результата выполнения.

Функция не хранит результатов предыдущих вызовов, локального кеша для быстрого возврата значения нет.

### Пример:

```
input int shift=0;
//+-----+
//| Function-event handler "tick" |
//+-----+
void OnTick()
{
    datetime time = iTime(Symbol(), Period(), shift);
    double open = iOpen(Symbol(), Period(), shift);
    double high = iHigh(Symbol(), Period(), shift);
    double low = iLow(Symbol(), Period(), shift);
    double close = iClose(NULL, PERIOD_CURRENT, shift);
```

```
long      volume= iVolume(Symbol(),0,shift);
int       bars   = iBars(NULL,0);

Comment(Symbol(),"","",EnumToString(Period()),"\n",
        "Time: " ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
        "Open: " ,DoubleToString(open,Digits()),"\n",
        "High: " ,DoubleToString(high,Digits()),"\n",
        "Low: " ,DoubleToString(low,Digits()),"\n",
        "Close: " ,DoubleToString(close,Digits()),"\n",
        "Volume: " ,IntegerToString(volume),"\n",
        "Bars: " ,IntegerToString(bars),"\n"
    );
}
```

Смотри также

[CopyOpen](#), [CopyRates](#)

## iTime

Возвращает значение времени открытия бара (указанного параметром shift) соответствующего графика.

```
datetime iTime(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    int               shift             // сдвиг
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

*shift*

[in] Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество баров назад).

### Возвращаемое значение

Значение времени открытия бара (указанного параметром shift) соответствующего графика или 0 в случае ошибки. Для получения дополнительной информации об [ошибке](#) необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция всегда возвращает актуальные данные, для этого она при каждом вызове делает запрос к таймсерии по указанным символу/периоду. Это означает, что при отсутствии готовых данных на первом вызове функции может понадобиться некоторое время для подготовки результата выполнения.

Функция не хранит результатов предыдущих вызовов, локального кеша для быстрого возврата значения нет

### Пример:

```
//-----
//| Script program start function
//+-----+
void OnStart()
{
//--- дата приходится на воскресенье
datetime time=D'2018.06.10 12:00';
string symbol="GBPUSD";
ENUM_TIMEFRAMES tf=PERIOD_H1;
bool exact=false;
//--- бара на указанное время нет - поэтому iBarShift вернет индекс ближайшего бара
```

```

int bar_index=iBarShift(symbol,tf,time,exact);
PrintFormat("1. %s %s %s(%s): bar index is %d (exact=%s)",
           symbol,EnumToString(tf),TimeToString(time),DayOfWeek(time),bar_index,exact);
datetime bar_time=iTime(symbol,tf,bar_index);
PrintFormat("Time of bar #%d is %s (%s)",
           bar_index,TimeToString(bar_time),DayOfWeek(bar_time));
//PrintFormat(iTime(symbol,tf,bar_index));
//--- требуем найти индекс бара на указанное время, но его нет - вернем -1
exact=true;
bar_index=iBarShift(symbol,tf,time,exact);
PrintFormat("2. %s %s %s (%s):bar index is %d (exact=%s)",
           symbol,EnumToString(tf),TimeToString(time),DayOfWeek(time),bar_index,exact);
}
//+-----+
//| Возвращает название дня недели |
//+-----+
string DayOfWeek(const datetime time)
{
    MqlDateTime dt;
    string day="";
    TimeToStruct(time,dt);
    switch(dt.day_of_week)
    {
        case 0: day=EnumToString(SUNDAY);
        break;
        case 1: day=EnumToString(MONDAY);
        break;
        case 2: day=EnumToString(TUESDAY);
        break;
        case 3: day=EnumToString(WEDNESDAY);
        break;
        case 4: day=EnumToString(THURSDAY);
        break;
        case 5: day=EnumToString(FRIDAY);
        break;
        default:day=EnumToString(SATURDAY);
        break;
    }
//---
    return day;
}
/* Результат:
1. GBPUSD PERIOD_H1 2018.06.10 12:00(SUNDAY): bar index is 64 (exact=false)
Time of bar #64 is 2018.06.08 23:00 (FRIDAY)
2. GBPUSD PERIOD_H1 2018.06.10 12:00 (SUNDAY):bar index is -1 (exact=true)
*/

```

**Смотри также**

[CopyTime](#), [CopyRates](#)

## iTickVolume

Возвращает значение тикового объема бара (указанного параметром shift) соответствующего графика.

```
long iTickVolume(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    int               shift             // сдвиг
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

*shift*

[in] Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество баров назад).

### Возвращаемое значение

Значение тикового объема бара (указанного параметром shift) соответствующего графика или 0 в случае ошибки. Для получения дополнительной информации об [ошибке](#) необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция всегда возвращает актуальные данные, для этого она при каждом вызове делает запрос к таймсерии по указанным символу/периоду. Это означает, что при отсутствии готовых данных на первом вызове функции может понадобиться некоторое время для подготовки результата выполнения.

Функция не хранит результатов предыдущих вызовов, локального кеша для быстрого возврата значения нет.

### Пример:

```
input int shift=0;
//+-----+
//| Function-event handler "tick" |
//+-----+
void OnTick()
{
    datetime time = iTime(Symbol(), Period(), shift);
    double open = iOpen(Symbol(), Period(), shift);
    double high = iHigh(Symbol(), Period(), shift);
    double low = iLow(Symbol(), Period(), shift);
    double close = iClose(NULL, PERIOD_CURRENT, shift);
```

```
long      volume= iVolume(Symbol(),0,shift);
int       bars   = iBars(NULL,0);

Comment(Symbol(),"","",EnumToString(Period()),"\n",
        "Time: " ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
        "Open: " ,DoubleToString(open,Digits()),"\n",
        "High: " ,DoubleToString(high,Digits()),"\n",
        "Low: " ,DoubleToString(low,Digits()),"\n",
        "Close: " ,DoubleToString(close,Digits()),"\n",
        "Volume: " ,IntegerToString(volume),"\n",
        "Bars: " ,IntegerToString(bars),"\n"
    );
}
```

#### Смотри также

[CopyTickVolume](#), [CopyRates](#)

## iRealVolume

Возвращает значение реального объема бара (указанного параметром shift) соответствующего графика.

```
long iRealVolume(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    int               shift             // сдвиг
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

*shift*

[in] Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество баров назад).

### Возвращаемое значение

Значение реального объема бара (указанного параметром shift) соответствующего графика или 0 в случае ошибки. Для получения дополнительной информации об [ошибке](#) необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция всегда возвращает актуальные данные, для этого она при каждом вызове делает запрос к таймсерии по указанным символу/периоду. Это означает, что при отсутствии готовых данных на первом вызове функции может понадобиться некоторое время для подготовки результата выполнения.

Функция не хранит результатов предыдущих вызовов, локального кеша для быстрого возврата значения нет.

### Пример:

```
input int shift=0;
//+-----+
//| Function-event handler "tick" |
//+-----+
void OnTick()
{
    datetime time = iTIME(Symbol(), Period(), shift);
    double open = iOpen(Symbol(), Period(), shift);
    double high = iHigh(Symbol(), Period(), shift);
    double low = iLow(Symbol(), Period(), shift);
    double close = iClose(NULL, PERIOD_CURRENT, shift);
```

```
long      volume= iVolume(Symbol(),0,shift);
int       bars   = iBars(NULL,0);

Comment(Symbol(),"","",EnumToString(Period()),"\n",
        "Time: " ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
        "Open: " ,DoubleToString(open,Digits()),"\n",
        "High: " ,DoubleToString(high,Digits()),"\n",
        "Low: " ,DoubleToString(low,Digits()),"\n",
        "Close: " ,DoubleToString(close,Digits()),"\n",
        "Volume: " ,IntegerToString(volume),"\n",
        "Bars: " ,IntegerToString(bars),"\n"
    );
}
```

#### Смотри также

[CopyRealVolume](#), [CopyRates](#)

## iVolume

Возвращает значение тикового объема бара (указанного параметром shift) соответствующего графика.

```
long iVolume(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES  timeframe,        // период
    int               shift             // сдвиг
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

*shift*

[in] Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество баров назад).

### Возвращаемое значение

Значение тикового объема бара (указанного параметром shift) соответствующего графика или 0 в случае ошибки. Для получения дополнительной информации об [ошибке](#) необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция всегда возвращает актуальные данные, для этого она при каждом вызове делает запрос к таймсерии по указанным символу/периоду. Это означает, что при отсутствии готовых данных на первом вызове функции может понадобиться некоторое время для подготовки результата выполнения.

Функция не хранит результатов предыдущих вызовов, локального кеша для быстрого возврата значения нет.

### Пример:

```
input int shift=0;
//+-----+
//| Function-event handler "tick" |
//+-----+
void OnTick()
{
    datetime time = iTIME(Symbol(), Period(), shift);
    double open = iOpen(Symbol(), Period(), shift);
    double high = iHigh(Symbol(), Period(), shift);
    double low = iLow(Symbol(), Period(), shift);
    double close = iClose(NULL, PERIOD_CURRENT, shift);
```

```
long      volume= iVolume(Symbol(),0,shift);
int       bars   = iBars(NULL,0);

Comment(Symbol(),"","",EnumToString(Period()),"\n",
        "Time: " ,TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
        "Open: " ,DoubleToString(open,Digits()),"\n",
        "High: " ,DoubleToString(high,Digits()),"\n",
        "Low: " ,DoubleToString(low,Digits()),"\n",
        "Close: " ,DoubleToString(close,Digits()),"\n",
        "Volume: " ,IntegerToString(volume),"\n",
        "Bars: " ,IntegerToString(bars),"\n"
    );
}
```

#### Смотри также

[CopyTickVolume](#), [CopyRates](#)

## iSpread

Возвращает значение спреда бара (указанного параметром shift) соответствующего графика.

```
long iSpread(
    const string      symbol,           // символ
    ENUM_TIMEFRAMES timeframe,        // период
    int               shift,            // сдвиг
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента. [NULL](#) означает текущий символ.

*timeframe*

[in] Период. Может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

*shift*

[in] Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество баров назад).

### Возвращаемое значение

Значение спреда для бара (указанного параметром shift) соответствующего графика или 0 в случае ошибки. Для получения дополнительной информации об [ошибке](#) необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция всегда возвращает актуальные данные, для этого она при каждом вызове делает запрос к таймсерии по указанным символу/периоду. Это означает, что при отсутствии готовых данных на первом вызове функции может понадобиться некоторое время для подготовки результата выполнения.

Функция не хранит результатов предыдущих вызовов, локального кеша для быстрого возврата значения нет.

### Пример:

```
input int shift=0;
//+-----+
//| Function-event handler "tick" |
//+-----+
void OnTick()
{
    datetime time = iTime(Symbol(), Period(), shift);
    double open = iOpen(Symbol(), Period(), shift);
    double high = iHigh(Symbol(), Period(), shift);
    double low = iLow(Symbol(), Period(), shift);
    double close = iClose(NULL, PERIOD_CURRENT, shift);
    long volume= iVolume(Symbol(), 0, shift);
```

```
int      bars   = iBars(NULL,0);

Comment(Symbol(),"","",EnumToString(Period()),"\n",
        "Time: " , TimeToString(time,TIME_DATE|TIME_SECONDS),"\n",
        "Open: " , DoubleToString(open,Digits()),"\n",
        "High: " , DoubleToString(high,Digits()),"\n",
        "Low: " , DoubleToString(low,Digits()),"\n",
        "Close: " , DoubleToString(close,Digits()),"\n",
        "Volume: " , IntegerToString(volume),"\n",
        "Bars: " , IntegerToString(bars),"\n"
    );
}
```

Смотри также

[CopySpread](#), [CopyRates](#)

## Пользовательские символы

Функции для создания и редактирования свойств пользовательских символов.

При подключении терминала к конкретному торговому серверу пользователь получает возможность [работать с таймсериями](#) тех финансовых инструментов, которые предоставляет данный брокер. Доступные финансовые инструменты показываются списком символов в окне Market Watch, отдельная группа функций позволяет [получать информацию о свойствах символа](#), торговых сессиях и обновлениях стакана заявок.

Представленная в этом разделе группа функций позволяет создавать свои собственные пользовательские символы. Для этого можно использовать существующие символы торгового сервера, текстовые файлы или внешние источники данных.

Функция	Действие
<a href="#">CustomSymbolCreate</a>	Создает пользовательский символ с указанным именем в указанной группе
<a href="#">CustomSymbolDelete</a>	Удаляет пользовательский символ с указанным именем
<a href="#">CustomSymbolSetInteger</a>	Устанавливает для пользовательского символа значение свойства целочисленного типа
<a href="#">CustomSymbolSetDouble</a>	Устанавливает для пользовательского символа значение свойства вещественного типа
<a href="#">CustomSymbolSetString</a>	Устанавливает для пользовательского символа значение свойства строкового типа
<a href="#">CustomSymbolSetMarginRate</a>	Устанавливает для пользовательского символа коэффициенты взимания маржи в зависимости от типа и направления ордера
<a href="#">CustomSymbolSetSessionQuote</a>	Устанавливает время начала и время окончания указанной котировочной сессии для указанных символа и дня недели
<a href="#">CustomSymbolSetSessionTrade</a>	Устанавливает время начала и время окончания указанной торговой сессии для указанных символа и дня недели
<a href="#">CustomRatesDelete</a>	Удаляет все бары в указанном временном интервале из ценовой истории пользовательского инструмента
<a href="#">CustomRatesReplace</a>	Полностью заменяет ценовую историю пользовательского инструмента в указанном временном интервале данными из массива типа MqlRates

<a href="#"><u>CustomRatesUpdate</u></a>	Добавляет в историю пользовательского инструмента отсутствующие бары и заменяет существующие бары данными из массива типа MqlRates
<a href="#"><u>CustomTicksAdd</u></a>	Добавляет в ценовую историю пользовательского инструмента данные из массива типа MqlTick. Пользовательский символ должен быть выбран в окне MarketWatch (Обзор рынка)
<a href="#"><u>CustomTicksDelete</u></a>	Удаляет все тики в указанном временном интервале из ценовой истории пользовательского инструмента
<a href="#"><u>CustomTicksReplace</u></a>	Полностью заменяет ценовую историю пользовательского инструмента в указанном временном интервале данными из массива типа MqlTick
<a href="#"><u>CustomBookAdd</u></a>	Передает состояние стакана цен по пользовательскому инструменту

## CustomSymbolCreate

Создает пользовательский символ с указанным именем в указанной группе.

```
bool CustomSymbolCreate(
    const string     symbol_name,           // имя пользовательского символа
    const string     symbol_path="",        // название группы, в которой будет создан си
    const string     symbol_origin=NULL    // имя символа, на основе которого будет созд
);
```

### Параметры

*symbol\_name*

[in] Имя пользовательского символа. Не должно содержать групп или подгрупп, в котором символ находится.

*symbol\_path=""*

[in] Имя группы, в которой создается символ.

*symbol\_origin=NULL*

[in] Имя символа, из которого будут скопированы [свойства](#) создаваемого пользовательского символа. После создания пользовательского символа можно изменить любое свойство на нужное значение соответствующими функциями.

### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Все пользовательские символы создаются в специальном разделе Custom. Если имя группы не задано (параметр *symbol\_path* в функции *CustomSymbolCreate* содержит пустую строку или NULL), то пользовательский символ будет создан в корне раздела Custom. Здесь можно провести аналогию с файловой системой, где группы-подгруппы могут рассматриваться как папки-подпапки

Имя символа и название группы задается только латинскими буквами без знаков препинания, пробелов и спецсимволов (допускаются ".", "\_", "&" и "#"). Не рекомендуется использовать в названии символы <, >, :, ", /, |, ?, \*.

Имя пользовательского символа должно быть уникальным независимо от названия группы, в котором он создается. Если символ с таким именем уже существует, то функция *CustomSymbolCreate()* вернёт false, а последующий вызов [GetLastError\(\)](#) выдаст код ошибки 5300 (ERR\_NOT\_CUSTOM\_SYMBOL) или 5304 (ERR\_CUSTOM\_SYMBOL\_EXIST).

Длина имени символа не должна превышать 31 знака, в противном случае *CustomSymbolCreate()* вернёт false и будет взведена ошибка 5302 - ERR\_CUSTOM\_SYMBOL\_NAME\_LONG.

Параметр *symbol\_path* допускается задавать двумя способами:

- только имя группы без имени пользовательского символа, например - "CFD\\Metals". Лучше всего использовать именно этот вариант, чтобы избегать ошибок.
- либо имя <группы> + разделитель групп "\\+<имя пользовательского символа>, например - "CFD\\Metals\\Platinum". В этом случае имя группы должно оканчиваться точным именем

пользовательского символа. В случае несовпадения пользовательский символ все равно будет создан, но не в той группе, которая задумывалась. Например, если `symbol_path="CFD\\Metals\\Platinum"` и `symbol_name="platinum"` (ошибка в регистре), то будет создан пользовательский символ с именем "platinum" в группе "Custom\CFD\Metals\Platinum". При этом функция `SymbolInfoGetString("platinum",SYMBOL_PATH)` вернёт значение "Custom\CFD\Metals\Platinum\platinum".

Необходимо иметь в виду, что свойство `SYMBOL_PATH` возвращает путь вместе с именем символа на конце. Поэтому нельзя просто так копировать его без изменений, если необходимо создать пользовательский символ в точно такой же группе. В этом случае необходимо отрезать имя символа, чтобы не получить результат, описанный выше.

Если в качестве параметра `symbol_origin` задан несуществующий символ, то пользовательский символ будет создан пустым, как если бы параметр `symbol_origin` не был указан. При этом будет взведена ошибка 4301 - `ERR_MARKET_UNKNOWN_SYMBOL`.

Длина параметра `symbol_path` не должна превышать 127 знаков с учетом "Custom\\", разделителей групп "\\\" и имени символа, если оно указано в конце.

#### Смотри также

[SymbolName](#), [SymbolSelect](#), [CustomSymbolDelete](#)

## CustomSymbolDelete

Удаляет пользовательский символ с указанным именем.

```
bool CustomSymbolDelete(
    const string      symbol_name           // имя пользовательского символа
);
```

### Параметры

*symbol*

[in] Имя пользовательского символа. Не должно совпадать с именем уже существующего символа.

### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Пользовательский символ, отображаемый в обзоре рынка (Market Watch) или по которому открыт график, не может быть удалён.

### Смотри также

[SymbolName](#), [SymbolSelect](#), [CustomSymbolCreate](#)

## CustomSymbolSetInteger

Устанавливает для пользовательского символа значение свойства целочисленного типа.

```
bool CustomSymbolSetInteger(
    const string          symbol_name,      // имя символа
    ENUM_SYMBOL_INFO_INTEGER property_id,   // идентификатор свойства
    long                  property_value    // значение свойства
);
```

### Параметры

*symbol\_name*

[in] Имя пользовательского символа.

*property\_id*

[in] Идентификатор свойства символа. Значение может быть одним из значений перечисления [ENUM\\_SYMBOL\\_INFO\\_INTEGER](#).

*property\_value*

[in] Переменная типа long, содержащая значение свойства.

### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Минутная и тиковая история пользовательского символа полностью удаляется, если в спецификации символа изменить любое из этих свойств:

- SYMBOL\_CHART\_MODE - тип цены, который используется для построения баров (Bid или Last)
- SYMBOL\_DIGITS - количество знаков после запятой для отражения цены

После удаления истории пользовательского символа терминал попытается создать новую историю с использованием обновленных свойств. То же самое происходит и при ручном изменении свойств пользовательского символа.

### Смотри также

[SymbolInfoInteger](#)

## CustomSymbolSetDouble

Устанавливает для пользовательского символа значение свойства вещественного типа.

```
bool CustomSymbolSetDouble(
    const string          symbol_name,      // имя символа
    ENUM_SYMBOL_INFO_DOUBLE property_id,    // идентификатор свойства
    double                property_value     // значение свойства
);
```

### Параметры

*symbol\_name*

[in] Имя пользовательского символа.

*property\_id*

[in] Идентификатор свойства символа. Значение может быть одним из значений перечисления [ENUM\\_SYMBOL\\_INFO\\_DOUBLE](#).

*property\_value*

[in] Переменная типа `double`, содержащая значение свойства.

### Возвращаемое значение

`true` - в случае успеха, иначе `false`. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Минутная и тиковая история пользовательского символа полностью удаляется, если в спецификации символа изменить любое из этих свойств:

- `SYMBOL_POINT` - значение одного пункта
- `SYMBOL_TRADE_TICK_SIZE` - значение одного тика, которое задает минимальное допустимое изменение цены
- `SYMBOL_TRADE_TICK_VALUE` - стоимость изменения цены в один тик для прибыльной позиции

После удаления истории пользовательского символа терминал попытается создать новую историю с использованием обновленных свойств. То же самое происходит и при ручном изменении свойств пользовательского символа.

### Смотри также

[SymbolInfoDouble](#)

## CustomSymbolSetString

Устанавливает для пользовательского символа значение свойства строкового типа.

```
bool CustomSymbolSetString(
    const string          symbol_name,      // имя символа
    ENUM_SYMBOL_INFO_STRING property_id,    // идентификатор свойства
    string                property_value     // значение свойства
);
```

### Параметры

*symbol\_name*

[in] Имя пользовательского символа.

*property\_id*

[in] Идентификатор свойства символа. Значение может быть одним из значений перечисления [ENUM\\_SYMBOL\\_INFO\\_STRING](#).

*property\_value*

[in] Переменная типа `string`, содержащая значение свойства.

### Возвращаемое значение

`true` - в случае успеха, иначе `false`. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Минутная и тиковая история пользовательского символа полностью удаляется, если в спецификации символа изменить свойство `SYMBOL_FORMULA`, которое задает формулу для построения цены пользовательского символа. После удаления истории пользовательского символа терминал попытается создать новую историю по новой формуле. То же самое происходит и при ручном изменении формулы пользовательского символа.

### Смотри также

[SymbolInfoString](#)

## CustomSymbolSetMarginRate

Устанавливает для пользовательского символа коэффициенты взимания маржи в зависимости от типа и направления ордера.

```
bool CustomSymbolSetMarginRate(
    const string      symbol_name,           // имя символа
    ENUM_ORDER_TYPE   order_type,            // тип ордера
    double            initial_margin_rate,    // коэффициент взимания начальной маржи
    double            maintenance_margin_rate // коэффициент взимания поддерживающей маржи
);
```

### Параметры

*symbol\_name*

[in] Имя пользовательского символа.

*order\_type*

[in] Тип ордера.

*initial\_margin\_rate*

[in] Переменная типа `double` со значением коэффициента взимания начальной маржи. Начальная маржа - это размер гарантийной суммы под совершение сделки объемом в 1 лот соответствующего направления. Умножая коэффициент на начальную маржу, мы можем получить размер средств, который будет зарезервирован на счете при размещении ордера указанного типа.

*maintenance\_margin\_rate*

[in] Переменная типа `double` со значением коэффициента взимания поддерживающей маржи. Поддерживающая маржа - это размер минимальной суммы для поддержания открытой позиции объемом в 1 лот соответствующего направления. Умножая коэффициент на поддерживающую маржу, мы можем получить размер средств, который будет зарезервирован на счете после срабатывания ордера указанного типа.

### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Смотри также

[SymbolInfoMarginRate](#)

## CustomSymbolSetSessionQuote

Устанавливает время начала и время окончания указанной котировочной сессии для указанных символа и дня недели.

```
bool CustomSymbolSetSessionQuote(
    const string      symbol_name,           // имя символа
    ENUM_DAY_OF_WEEK day_of_week,           // день недели
    uint              session_index,         // номер сессии
    datetime          from,                 // время начала сессии
    datetime          to                    // время окончания сессии
);
```

### Параметры

*symbol\_name*

[in] Имя пользовательского символа.

*ENUM\_DAY\_OF\_WEEK*

[in] День недели, значение из перечисления [ENUM\\_DAY\\_OF\\_WEEK](#).

*uint*

[in] Порядковый номер сессии, для которой нужно установить время начала и время окончания. Индексация сессий начинается с 0.

*from*

[in] Время начала сессии в секундах от 00 часов 00 минут, значение даты в переменной будет проигнорировано.

*to*

[in] Время окончания сессии в секундах от 00 часов 00 минут, значение даты в переменной будет проигнорировано.

### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Если сессия с указанным *session\_index* уже существует, то функция просто отредактирует начало и конец сессии.

Если для сессии переданы нулевые параметры начала и конца, то есть указаны *from=0* и *to=0*, то соответствующая сессия с индексом *session\_index* удаляется, а сама нумерация сессий сдвигается вниз.

Добавлять сессии можно только последовательно, то есть сессию с индексом *session\_index=1* можно добавить только в том случае, если уже существует сессия с индексом равным 0. При нарушении этого правила новая сессия не создается, а сама функция вернет значение false.

### Смотри также

[SymbolInfoSessionQuote](#), [Информация об инструменте](#), [TimeToStruct](#), [Структура даты](#)

## CustomSymbolSetSessionTrade

Устанавливает время начала и время окончания указанной торговой сессии для указанных символа и дня недели.

```
bool CustomSymbolSetSessionTrade(
    const string      symbol_name,           // имя символа
    ENUM_DAY_OF_WEEK day_of_week,            // день недели
    uint              session_index,          // номер сессии
    datetime          from,                  // время начала сессии
    datetime          to                     // время окончания сессии
);
```

### Параметры

*symbol\_name*

[in] Имя пользовательского символа.

*ENUM\_DAY\_OF\_WEEK*

[in] День недели, значение из перечисления [ENUM\\_DAY\\_OF\\_WEEK](#).

*uint*

[in] Порядковый номер сессии, для которой нужно установить время начала и время окончания. Индексация сессий начинается с 0.

*from*

[in] Время начала сессии в секундах от 00 часов 00 минут, значение даты в переменной будет проигнорировано.

*to*

[in] Время окончания сессии в секундах от 00 часов 00 минут, значение даты в переменной будет проигнорировано.

### Возвращаемое значение

true - в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Если сессия с указанным *session\_index* уже существует, то функция просто отредактирует начало и конец сессии.

Если для сессии переданы нулевые параметры начала и конца, то есть указаны *from=0* и *to=0*, то соответствующая сессия с индексом *session\_index* удаляется, а сама нумерация сессий сдвигается вниз.

Добавлять сессии можно только последовательно, то есть сессию с индексом *session\_index=1* можно добавить только в том случае, если уже существует сессия с индексом равным 0. При нарушении этого правила новая сессия не создается, а сама функция вернет значение false.

### Смотри также

[SymbolInfoSessionTrade](#), [Информация об инструменте](#), [TimeToStruct](#), [Структура даты](#)

## CustomRatesDelete

Удаляет все бары в указанном временном интервале из ценовой истории пользовательского инструмента.

```
int CustomRatesDelete(
    const string      symbol,           // имя символа
    datetime         from,             // с какой даты
    datetime         to,               // по какую дату
);
```

### Параметры

*symbol*

[in] Имя пользовательского инструмента.

*from*

[in] Время первого бара в ценовой истории из указанного диапазона, подлежащего удалению.

*to*

[in] Время последнего бара в ценовой истории из указанного диапазона, подлежащего удалению.

### Возвращаемое значение

Количество удаленных баров либо -1 в случае [ошибки](#).

### Смотри также

[CustomRatesReplace](#), [CustomRatesUpdate](#), [CopyRates](#)

## CustomRatesReplace

Полностью заменяет ценовую историю пользовательского инструмента в указанном временном интервале данными из массива типа [MqlRates](#).

```
int CustomRatesReplace(
    const string      symbol,           // имя символа
    datetime         from,             // с какой даты
    datetime         to,               // по какую дату
    const MqlRates& rates[],          // массив с данными, которые необходимо применить
    uint             count=WHOLE_ARRAY // количество элементов, которые будут использованы
);
```

### Параметры

*symbol*

[in] Имя пользовательского инструмента.

*from*

[in] Время первого бара в ценовой истории из указанного диапазона, подлежащего обновлению.

*to*

[in] Время последнего бара в ценовой истории из указанного диапазона, подлежащего обновлению.

*rates[]*

[in] Массив исторических данных типа [MqlRates](#) для таймфрейма M1.

*count=WHOLE\_ARRAY*

[in] Количество элементов из массива *rates[]*, которые будут использованы для замены. Значение [WHOLE\\_ARRAY](#) означает, что для замены необходимо использовать все элементы массива *rates[]*.

### Возвращаемое значение

Количество обновленных баров либо -1 в случае [ошибки](#).

### Примечание

Если бар из массива *rates[]* выходит за пределы указанного диапазона, то он игнорируется. Если такой бар уже есть в ценовой истории и входит в заданный диапазон, то он заменяется. Все остальные бары в текущей ценовой истории за пределами указанного диапазона остаются неизменным. Данные в массиве *rates[]* должны быть корректными по ценам OHLC, а время открытия баров соответствовать [таймфрейму M1](#).

### Смотри также

[CustomRatesDelete](#), [CustomRatesUpdate](#), [CopyRates](#)

## CustomRatesUpdate

Добавляет в историю пользовательского инструмента отсутствующие бары и заменяет существующие данными из массива типа [MqlRates](#).

```
int CustomRatesUpdate(
    const string      symbol,           // имя пользовательского символа
    const MqlRates&  rates[],          // массив с данными, которые необходимо применить
    uint              count=WHOLE_ARRAY // количество элементов, которые будут использованы
);
```

### Параметры

*symbol*

[in] Имя пользовательского инструмента.

*rates[]*

[in] Массив исторических данных типа [MqlRates](#) для таймфрейма M1.

*count=WHOLE\_ARRAY*

[in] Количество элементов из массива *rates[]*, которые будут использованы для обновления.

Значение [WHOLE\\_ARRAY](#) означает, что необходимо использовать все элементы массива *rates[]*.

### Возвращаемое значение

Количество обновленных баров либо -1 в случае [ошибки](#).

### Примечание

Если бар из массива *rates[]* отсутствует в текущей истории пользовательского инструмента, то он добавляется. Если такой бар уже есть, то он заменяется. Все остальные бары в текущей ценовой истории остаются неизменным. Данные в массиве *rates[]* должны быть корректными по ценам OHLC, а время открытия баров соответствовать [таймфрейму](#) M1.

### Смотри также

[CustomRatesReplace](#), [CustomRatesDelete](#), [CopyRates](#)

## CustomTicksAdd

Добавляет в ценовую историю пользовательского инструмента данные из массива типа [MqlTick](#). Пользовательский символ должен быть выбран в окне MarketWatch (Обзор рынка).

```
int CustomTicksAdd(
    const string      symbol,           // имя символа
    const MqlTick&   ticks[],          // массив с тиковыми данными, которые необходимо добавить
    uint              count=WHOLE_ARRAY // количество элементов, которые будут использованы
);
```

### Параметры

*symbol*

[in] Имя пользовательского инструмента.

*ticks[]*

[in] Массив тиковых данных типа [MqlTick](#), упорядоченных по времени в порядке возрастания, то есть требуется чтобы  $\text{ticks}[k].time\_msc \leq \text{ticks}[n].time\_msc$ , если  $k < n$ .

*count=WHOLE\_ARRAY*

[in] Количество элементов из массива *ticks[]*, которые будут использованы для добавления. Значение [WHOLE\\_ARRAY](#) означает, что необходимо использовать все элементы массива *ticks[]*.

### Возвращаемое значение

Количество добавленных тиков либо -1 в случае ошибки.

### Примечание

Функция [CustomTicksAdd](#) работает только для пользовательских символов, открытых в окне MarketWatch (Обзор рынка). Если символ не выбран в MarketWatch, то для вставки тиков необходимо использовать [CustomTicksReplace](#).

Функция [CustomTicksAdd](#) позволяет транслировать тики так, как если бы они приходили от сервера брокера. Данные записываются не напрямую в базу тиков, а отправляются в окно "Обзор рынка". И уже из него терминал сохраняет тики в своей базе. При большом объеме данных, передаваемых за один вызов, функция меняет свое поведение для экономии ресурсов. Если передается более 256 тиков, данные делятся на две части. Первая часть (большая) сразу напрямую записывается в базу тиков (как это делает [CustomTicksReplace](#)). Вторая часть, состоящая из последних 128 тиков, передается в окно "Обзор рынка" и после этого сохраняется терминалом в базе.

Структура [MqlTick](#) имеет два поля со значением времени - *time* (время тика в секундах) и *time\_msc* (время тика в миллисекундах) - которые ведут отсчет от 01 января 1970 года. Обработка этих полей в добавляемых тиках производится по следующим правилам в указанном порядке:

- если значение  $\text{ticks}[k].time\_msc \neq 0$ , то используем его для заполнения поля *ticks[k].time*, то есть для тика выставляется время  $\text{ticks}[k].time = \text{ticks}[k].time\_msc / 1000$  (деление целочисленное)
- если  $\text{ticks}[k].time\_msc = 0$  и  $\text{ticks}[k].time \neq 0$ , то время в миллисекундах получается умножением на 1000, то есть  $\text{ticks}[k].time\_msc = \text{ticks}[k].time * 1000$

3. если ticks[k].time\_msc==0 и ticks[k].time==0, то в эти поля записывается текущее [время торгового сервера](#) с точностью до миллисекунд на момент вызова функции CustomTicksAdd.

Если значение полей ticks[k].bid, ticks[k].ask, ticks[k].last или ticks[k].volume больше нуля, то в поле ticks[k].flags пишется комбинация соответствующих флагов:

- TICK\_FLAG\_BID - тик изменил цену бид
- TICK\_FLAG\_ASK - тик изменил цену аск
- TICK\_FLAG\_LAST - тик изменил цену последней сделки
- TICK\_FLAG\_VOLUME - тик изменил объем

Если значение какого-то поля меньше или равно нуля, соответствующий ему флаг не записываются в поле ticks[k].flags.

Флаги TICK\_FLAG\_BUY и TICK\_FLAG\_SELL в историю пользовательского инструмента не добавляются.

#### Смотри также

[CustomRatesDelete](#), [CustomRatesUpdate](#), [CustomTicksReplace](#), [CopyTicks](#), [CopyTicksRange](#)

## CustomTicksDelete

Удаляет все тики в указанном временном интервале из ценовой истории пользовательского инструмента.

```
int CustomTicksDelete(
    const string      symbol,           // имя символа
    long              from_msc,         // с какой даты
    long              to_msc            // по какую дату
);
```

### Параметры

*symbol*

[in] Имя пользовательского инструмента.

*from\_msc*

[in] Время первого тика в ценовой истории из указанного диапазона, подлежащего удалению.  
Время в миллисекундах с 01.01.1970.

*to\_msc*

[in] Время последнего тика в ценовой истории из указанного диапазона, подлежащего удалению.  
Время в миллисекундах с 01.01.1970.

### Возвращаемое значение

Количество удаленных тиков либо -1 в случае [ошибки](#).

### Смотри также

[CustomRatesDelete](#), [CustomRatesUpdate](#), [CustomTicksReplace](#), [CopyTicks](#), [CopyTicksRange](#)

## CustomTicksReplace

Полностью заменяет ценовую историю пользовательского инструмента в указанном временном интервале данными из массива типа [MqlTick](#).

```
int CustomTicksReplace(
    const string      symbol,           // имя символа
    long              from_msc,         // с какой даты
    long              to_msc,           // по какую дату
    const MqlTick&   ticks[],          // массив с тиковыми данными, которые необходимы
    uint              count=WHOLE_ARRAY // количество элементов, которые будут использованы
);
```

### Параметры

*symbol*

[in] Имя пользовательского инструмента.

*from\_msc*

[in] Время первого тика в ценовой истории из указанного диапазона, подлежащего удалению. Время в миллисекундах с 01.01.1970.

*to\_msc*

[in] Время последнего тика в ценовой истории из указанного диапазона, подлежащего удалению. Время в миллисекундах с 01.01.1970.

*ticks[]*

[in] Массив тиковых данных типа [MqlTick](#), упорядоченных по времени в порядке возрастания.

*count=WHOLE\_ARRAY*

[in] Количество элементов из массива *ticks[]*, которые будут использованы для замены в указанном интервале времени. Значение [WHOLE\\_ARRAY](#) означает, что необходимо использовать все элементы массива *ticks[]*.

### Возвращаемое значение

Количество обновленных тиков либо -1 в случае [ошибки](#).

### Примечание

Так как в потоке котировок нередко несколько тиков могут иметь одно и то же время с точностью до миллисекунды (точное время тика хранится в поле *time\_msc* структуры [MqlTick](#)), то функция [CustomTicksReplace](#) не делает автоматической сортировки элементов массива *ticks[]* по времени. Поэтому массив тиков должен быть предварительно упорядочен по возрастанию времени.

Замена тиков производится последовательно день за днём до времени указанного в *to\_msc* либо до момента возникновения ошибки. Сначала обрабатывается первый день из указанного диапазона, затем следующий, и так далее. Как только обнаружится несоответствие времени тика порядку возрастания (неубывания), то процесс замены тиков сразу же прекращается на текущем дне. При этом тики за предыдущие дни будут успешно заменены, а текущий день (на момент неправильного тика) и все оставшиеся дни в указанном интервале останутся без изменения.

Если в массиве `ticks[]` отсутствуют тиковые данные за какой-то день (вообще говоря, интервал любой длительности), то после применения тиковых данных из `ticks[]` в истории пользователяского инструмента образуется "дыра", соответствующая пропущенным данным. Другими словами, вызов [CustomTicksReplace](#) с вырезанными тиками за конкретный интервал будет равносильно удалению части тиковой истории как будто вызывается [CustomTicksDelete](#) с интервалом "дыры".

Если в базе тиков в указанном интервале времени данные отсутствуют, то `CustomTicksReplace` просто добавит в нее тики из массива `ticks[]`.

Функция `CustomTicksReplace` работает напрямую с базой данных тиков.

#### Смотри также

[CustomRatesDelete](#), [CustomRatesUpdate](#), [CustomTicksDelete](#), [CopyTicks](#), [CopyTicksRange](#)

## CustomBookAdd

Передает состояние стакана цен по пользовательскому инструменту. Функция позволяет транслировать стакан цен так, как если бы он приходил от сервера брокера.

```
int CustomBookAdd(
    const string      symbol,           // имя символа
    const MqlBookInfo& books[],        // массив с описаниями элементов стакана цен
    uint              count=WHOLE_ARRAY // количество элементов, которые будут добавлены
);
```

### Параметры

*symbol*

[in] Имя пользовательского инструмента.

*books[]*

[in] Массив данных типа [MqlBookInfo](#), полностью описывающих состояние стакана цен — все заявки на покупку и продажу. Переданное состояние стакана цен полностью заменяет предыдущее.

*count=WHOLE\_ARRAY*

[in] Количество элементов массива *books[]*, которое должно быть передано в функцию. По умолчанию используется весь массив.

### Возвращаемое значение

Количество добавленных тиков либо -1 в случае [ошибки](#).

### Примечание

Функция [CustomBookAdd](#) работает только для пользовательских символов, по которым открыт стакан цен — через интерфейс платформы или функцию [MarketBookAdd](#).

При вбросе стакана цен Bid и Ask инструмента не обновляются. Вы должны самостоятельно контролировать изменение лучших цен и вбрасывать тики при помощи [CustomTicksAdd](#).

Передаваемые данные проверяются на корректность: цены и объемы не должны быть отрицательными, для каждого элемента должны быть указаны тип, цена и объем ([MqlBookInfo.volume](#) или [MqlBookInfo.volume\\_real](#)). Если хотя бы один элемент стакана описан неверно, система отбросит переданное состояние полностью.

Объем с повышенной точностью [MqlBookInfo.volume\\_real](#) имеет больший приоритет по сравнению с обычным [MqlBookInfo.volume](#). Если для элемента стакана указаны оба значения, будет использовано [volume\\_real](#).

Порядок следования элементов [MqlBookInfo](#) в массиве *books* не имеет значения. При сохранении данных терминал сортирует их по цене самостоятельно.

При сохранении данных проверяется параметр "Глубина стакана" ([SYMBOL\\_TICKS\\_BOOKDEPTH](#)) принимающего пользовательского инструмента. Если количество заявок на продажу в передаваемом стакане цен превышает это значение, лишние уровни отбрасываются. Аналогично для заявок на покупку.

Пример заполнения массива *books*:

Состояние стакана цен	Заполнение books[]																		
<table border="1"> <thead> <tr> <th>Volume</th><th>Price</th></tr> </thead> <tbody> <tr><td>100.00</td><td>1.14337</td></tr> <tr><td>50.00</td><td>1.14336</td></tr> <tr><td>40.00</td><td>1.14335</td></tr> <tr><td>10.00</td><td>1.14333</td></tr> <tr><td>10.00</td><td>1.14322</td></tr> <tr><td>90.00</td><td>1.14320</td></tr> <tr><td>100.00</td><td>1.14319</td></tr> <tr><td>10.00</td><td>1.14318</td></tr> </tbody> </table>	Volume	Price	100.00	1.14337	50.00	1.14336	40.00	1.14335	10.00	1.14333	10.00	1.14322	90.00	1.14320	100.00	1.14319	10.00	1.14318	<pre> books[0].type=BOOK_TYPE_SELL; books[0].price=1.14337; books[0].volume=100; books[1].type=BOOK_TYPE_SELL; books[1].price=1.14330; books[1].volume=50; books[2].type=BOOK_TYPE_SELL; books[2].price=1.14335; books[2].volume=40; books[3].type=BOOK_TYPE_SELL; books[3].price=1.14333; books[3].volume=10; books[4].type=BOOK_TYPE_BUY; books[4].price=1.14322; books[4].volume=10; books[5].type=BOOK_TYPE_BUY; books[5].price=1.14320; books[5].volume=90; books[6].type=BOOK_TYPE_BUY; books[6].price=1.14319; books[6].volume=100; books[7].type=BOOK_TYPE_BUY; books[7].price=1.14318; books[7].volume=10; </pre>
Volume	Price																		
100.00	1.14337																		
50.00	1.14336																		
40.00	1.14335																		
10.00	1.14333																		
10.00	1.14322																		
90.00	1.14320																		
100.00	1.14319																		
10.00	1.14318																		

Пример:

```

//+-----+
//| Expert initialization function           |
//+-----+
int OnInit()
{
//--- включаем стакан цен для инструмента, из которого будем брать данные
    MarketBookAdd(Symbol());
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function          |
//+-----+
void OnDeinit(const int reason)
{
}
//+-----+
//| Tick function                            |
//+-----+
void OnTick(void)
{
    MqlTick ticks[];
    ArrayResize(ticks,1);
}

```

```
//--- скопируем текущие цены из обычного инструмента в пользовательский
if(SymbolInfoTick(Symbol(),ticks[0]))
{
    string symbol_name=Symbol()+"._SYN";
    CustomTicksAdd(symbol_name,ticks);
}
//+-----+
//| Book function
//+-----+
void OnBookEvent(const string &book_symbol)
{
//--- скопируем текущее состояние стакана цен из обычного инструмента в пользовательск
if(book_symbol==Symbol())
{
    MqlBookInfo book_array[];
    if(MarketBookGet(Symbol(),book_array))
    {
        string symbol_name=Symbol()+"._SYN";
        CustomBookAdd(symbol_name,book_array);
    }
}
//+-----+
```

#### Смотри также

[MarketBookAdd](#), [CustomTicksAdd](#), [OnBookEvent](#)

## Операции с графиками

Функции для установки свойств графика ([ChartSetInteger](#), [ChartSetDouble](#), [ChartSetString](#)) являются асинхронными и служат для отправки графику команд на изменение. При успешном выполнении этих функций команда попадает в общую очередь событий графика. Изменение свойств графика производится в процессе обработки очереди событий данного графика.

По этой причине не следует ожидать немедленного обновления графика после вызова асинхронных функций. Для принудительного обновления внешнего вида и свойств графика используйте функцию [ChartRedraw\(\)](#).

Функция	Действие
<a href="#">ChartApplyTemplate</a>	Применяет к указанному графику шаблон из указанного файла
<a href="#">ChartSaveTemplate</a>	Сохраняет текущие настройки графика в шаблон с указанным именем
<a href="#">ChartWindowFind</a>	Возвращает номер подокна, в котором находится индикатор
<a href="#">ChartTimePriceToXY</a>	Преобразует координаты графика из представления время/цена в координаты по оси X и Y
<a href="#">ChartXYToTimePrice</a>	Преобразует координаты X и Y графика в значения время и цена
<a href="#">ChartOpen</a>	Открывает новый график с указанным символом и периодом
<a href="#">ChartClose</a>	Закрывает указанный график
<a href="#">ChartFirst</a>	Возвращает идентификатор первого графика клиентского терминала
<a href="#">ChartNext</a>	Возвращает идентификатор графика, следующего за указанным
<a href="#">ChartSymbol</a>	Возвращает имя символа указанного графика
<a href="#">ChartPeriod</a>	Возвращает значение периода указанного графика
<a href="#">ChartRedraw</a>	Вызывает принудительную перерисовку указанного графика
<a href="#">ChartSetDouble</a>	Задает значение типа <code>double</code> соответствующего свойства указанного графика
<a href="#">ChartSetInteger</a>	Задает значение целочисленного типа ( <code>datetime</code> , <code>int</code> , <code>color</code> , <code>bool</code> или <code>char</code> ) соответствующего свойства указанного графика

<a href="#"><u>ChartSetString</u></a>	Задает значение типа string соответствующего свойства указанного графика
<a href="#"><u>ChartGetDouble</u></a>	Возвращает значение соответствующего свойства указанного графика
<a href="#"><u>ChartGetInteger</u></a>	Возвращает целочисленное значение соответствующего свойства указанного графика
<a href="#"><u>ChartGetString</u></a>	Возвращает строковое значение соответствующего свойства указанного графика
<a href="#"><u>ChartNavigate</u></a>	Осуществляет сдвиг указанного графика на указанное количество баров относительно указанной позиции графика
<a href="#"><u>ChartID</u></a>	Возвращает идентификатор текущего графика
<a href="#"><u>ChartIndicatorAdd</u></a>	Добавляет на указанное окно графика индикатор с указанным хэндлом.
<a href="#"><u>ChartIndicatorDelete</u></a>	Удаляет с указанного окна графика индикатор с указанным именем
<a href="#"><u>ChartIndicatorGet</u></a>	Возвращает хэндл индикатора с указанным коротким именем на указанном окне графика
<a href="#"><u>ChartIndicatorName</u></a>	Возвращает короткое имя индикатора по номеру в списке индикаторов на указанном окне графика.
<a href="#"><u>ChartIndicatorsTotal</u></a>	Возвращает количество всех индикаторов, присоединенных к указанному окну графика.
<a href="#"><u>ChartWindowOnDropped</u></a>	Возвращает номер подокна графика, на которое брошен мышкой данный эксперт, скрипт, объект или индикатор
<a href="#"><u>ChartPriceOnDropped</u></a>	Возвращает ценовую координату, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт
<a href="#"><u>ChartTimeOnDropped</u></a>	Возвращает временную координату, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт
<a href="#"><u>ChartXOnDropped</u></a>	Возвращает координату по оси X, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт
<a href="#"><u>ChartYOnDropped</u></a>	Возвращает координату по оси Y, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт

<a href="#"><u>ChartSetSymbolPeriod</u></a>	Меняет значения символа и периода указанного графика
<a href="#"><u>ChartScreenShot</u></a>	Делает снимок указанного графика в формате GIF, PNG или BMP в зависимости от указанного расширения

## ChartApplyTemplate

Применяет к графику указанный шаблон. Отданная команда поступает в очередь сообщений графика и выполняется только после обработки всех предыдущих команд.

```
bool ChartApplyTemplate(
    long      chart_id,      // идентификатор графика
    const string filename     // имя файла с шаблоном
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*filename*

[in] Имя файла, содержащего шаблон.

### Возвращаемое значение

Возвращает true в случае удачного помещения команды в очередь графика, иначе false. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Если посредством этой функции из эксперта будет загружен новый шаблон на график, к которому он присоединен, то эксперт будет выгружен и не сможет продолжить работу.

В целях безопасности права на торговлю при применении шаблона к графику могут ограничиваться:

Права на торговлю не могут быть повышены при запуске советника путем применения шаблона с помощью функции `ChartApplyTemplate()`.

Если у MQL5-программы, которая вызывает функцию `ChartApplyTemplate()`, отсутствуют права на торговлю, то эксперт, загруженный при помощи шаблона, также не будет иметь прав на торговлю вне зависимости от настроек шаблона.

Если у MQL5-программы, которая вызывает функцию `ChartApplyTemplate()`, есть права на торговлю, а в настройках шаблона права отсутствуют, то советник, загруженный при помощи шаблона, не будет иметь прав на торговлю.

## Использование шаблонов

Средствами языка MQL5 можно задавать множество свойств графика, в том числе устанавливать цвета с помощью функции [ChartSetInteger\(\)](#) :

- Цвет фона графика;
- Цвет осей, шкалы и строки OHLC;
- Цвет сетки;
- Цвет объемов и уровней открытия позиций;
- Цвет бара вверх, тени и окантовки тела бычьей свечи;

- Цвет бара вниз, тени и окантовки тела медвежьей свечи;
- Цвет линии графика и японских свечей "Доджи";
- Цвет тела бычьей свечи;
- Цвет тела медвежьей свечи;
- Цвет линии Bid-цены;
- Цвет линии Ask-цены;
- Цвет линии цены последней совершенной сделки (Last);
- Цвет уровней стоп-ордеров (Stop Loss и Take Profit).

Кроме того, на графике может быть множество [графических объектов](#) и [индикаторов](#). Достаточно один раз настроить внешний вид графика со всеми необходимыми индикаторами и сохранить его в виде шаблона, чтобы в последствии можно было применять этот шаблон к любому графику.

Функция [ChartApplyTemplate\(\)](#) предназначена для использования ранее сохраненного шаблона и может использоваться в любой mql5-программе. Вторым параметром функции ChartApplyTemplate() передается путь к имени файла, в котором находится шаблон. Поиск файла шаблона осуществляется по следующим правилам:

- если в начале пути стоит разделитель обратная косая черта "\", (пишется "\\\"), то шаблон ищется относительно пути каталог\_данных\_терминала\MQL5,
- если обратной косой черты нет, то шаблон ищется относительно исполняемого EX5-файла, в котором происходит вызов функции ChartApplyTemplate();
- если шаблон не найден в первых двух вариантах, то поиск ведется в папке каталог\_терминала\Profiles\Templates\.

Здесь каталог\_терминала означает папку, из которой запущен клиентский терминал MetaTrader 5, а каталог\_данных\_терминала означает папку, в которой хранятся изменяемые файлы и ее расположение может зависеть от типа операционной системы, имени пользователя и настроек безопасности компьютера. В общем случае это разные папки, хотя в некоторых случаях могут и совпадать.

Расположение папок каталог\_данных\_терминала и каталог\_терминала можно узнать с помощью функции [TerminalInfoString\(\)](#).

```
//--- каталог из которой запущен терминал
string terminal_path=TerminalInfoString(TERMINAL_PATH);
Print("Каталог терминала:",terminal_path);
//--- каталог данных терминала, в котором находится папка MQL5 с советниками и индикаторами
string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
Print("Каталог данных терминала:",terminal_data_path);
```

Примеры записи:

```
//--- шаблон ищем в папке каталог_данных_терминала\MQL5\
ChartApplyTemplate(0,"\\first_template.tpl"))
//--- шаблон ищем в папке каталог_исполняемого_EX5_файла\, затем в папке каталог_данных_терминала
ChartApplyTemplate(0,"second_template.tpl"))
//--- шаблон ищем в папке каталог_исполняемого_EX5_файла\My_templates\, затем в папке каталог_данных_терминала
ChartApplyTemplate(0,"My_templates\\third_template.tpl"))
```

Шаблоны не относятся к ресурсам, их нельзя включать в исполняемый файл EX5.

**Пример:**

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- пример применения шаблона, расположенного в каталоге \MQL5\Files
if(FileIsExist("my_template.tpl"))
{
    Print("Шаблон my_template.tpl найден в каталоге \Files'");
//--- применим найденный шаблон
if(ChartApplyTemplate(0, "\\Files\\my_template.tpl"))
{
    Print("Применили успешно шаблон 'my_template.tpl'");
//--- принудительно перерисуем график для быстрого показа изменений
    ChartRedraw();
}
else
    Print("Не удалось применить шаблон 'my_template.tpl', ошибка ", GetLastError());
}
else
{
    Print("Файл 'my_template.tpl' не найден в папке "
        +TerminalInfoString(TERMINAL_PATH)+"\\MQL5\\Files");
}
}
```

**Смотри также**[Ресурсы](#)

## ChartSaveTemplate

Сохраняет текущие настройки графика в шаблон с указанным именем.

```
bool ChartSaveTemplate(
    long      chart_id,      // идентификатор графика
    const string filename     // имя файла для сохранения шаблона
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*filename*

[in] Имя файла для сохранения шаблона. Расширение ".tpl" будет добавлено к имени файла автоматически, указывать его не требуется. Шаблон сохраняется в папку **каталог\_данных\Profiles\Templates\** и может быть использован также и для ручного применения в терминале. Если шаблон с данным именем уже существует, то его содержимое будет переписано заново.

### Возвращаемое значение

При успешном выполнении функция возвращает `true`, в противном случае возвращает `false`. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Шаблон позволяет сохранить настройки графика со всеми наложенными на него индикаторами и графическими объектами, чтобы затем применить его на другом графике.

### Пример:

```
//+-----+
//|                               Test_ChartSaveTemplate.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input string          symbol="GBPUSD"; // символ нового графика
input ENUM_TIMEFRAMES period=PERIOD_H3; // таймфрейм нового графика
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- сначала набросим на график индикаторы
    int handle;
```

```

//--- подготовим индикатор к использованию
if(!PrepareZigzag(NULL,0,handle)) return; // попытка не удалась, выходим
//--- набросим индикатор на текущий график, но в отдельное окно
if(!ChartIndicatorAdd(0,1,handle))
{
    PrintFormat("Не удалось добавить на график %s/%s индикатор с хэндлом=%d. Код ошибки=%d",
               _Symbol,
               EnumToString(_Period),
               handle,
               GetLastError());
    //--- досрочно прекращаем работу программы
    return;
}
//--- прикажем графику обновиться, чтобы сразу увидеть наброшенный индикатор
ChartRedraw();
//--- найдем два последних перелома зигзага
double two_values[];
datetime two_times[];
if(!GetLastTwoFractures(two_values,two_times,handle))
{
    PrintFormat("Не удалось найти два последних перелома в индикаторе Zigzag!");
    //--- досрочно прекращаем работу программы
    return;
}
//--- теперь набросим канал стандартного отклонения
string channel="StdDeviation Channel";
if(!ObjectCreate(0,channel,OBJ_STDDEVCHANNEL,0,two_times[1],0))
{
    PrintFormat("Не удалось создать объект %s. Код ошибки %d",
               EnumToString(OBJ_STDDEVCHANNEL),GetLastError());
    return;
}
else
{
    //--- канал создан, доопределим вторую опорную точку
    ObjectSetInteger(0,channel,OBJPROP_TIME,1,two_times[0]);
    //--- зададим каналу текст всплывающей подсказки
    ObjectSetString(0,channel,OBJPROP_TOOLTIP,"Demo from MQL5 Help");
    //--- обновим график
    ChartRedraw();
}
//--- сохраним полученный результат в шаблон
ChartSaveTemplate(0,"StdDevChannelOnZigzag");
//--- откроем новый график и набросим на него сохраненный шаблон
long new_chart=ChartOpen(symbol,period);
//--- включим показ всплывающих подсказок для графических объектов
ChartSetInteger(new_chart,CHART_SHOW_OBJECT_DESCR,true);
if(new_chart!=0)
{

```

```

//--- набросим на новый график сохраненный шаблон
ChartApplyTemplate(new_chart,"StdDevChannelOnZigzag");
}

Sleep(10000);
}

//+-----+
//| Создает хэндл зигзага и обеспечивает готовность его данных |
//+-----+

bool PrepareZigzag(string sym,ENUM_TIMEFRAMES tf,int &h)
{
    ResetLastError();

//--- индикатор Zigzag должен находиться в папке каталог_данных_терминала\MQL5\Examples\
    h=iCustom(sym,tf,"Examples\\Zigzag");
    if(h==INVALID_HANDLE)
    {
        PrintFormat("%s: Не удалось создать хэндл индикатора Zigzag. Код ошибки %d",
                   __FUNCTION__,GetLastError());
        return false;
    }

//--- при создании хэндла индикатора ему требуется время на расчет значений
    int k=0; // количество попыток чтобы дождаться расчета индикатора
//--- ждем расчета в цикле, делаем паузу в 50 миллисекунд, если расчет еще не готов
    while(BarsCalculated(h)<=0)
    {
        k++;
        //--- выведем количество попыток
        PrintFormat("%s: k=%d",__FUNCTION__,k);
        //--- подождем 50 миллисекунд, пока индикатор рассчитается
        Sleep(50);

        //--- если сделано больше 100 попыток, то что-то не так
        if(k>100)
        {
            //--- сообщим о проблеме
            PrintFormat("Не удалось рассчитать индикатор за %d попыток!");
            //--- досрочно прекращаем работу программы
            return false;
        }
    }

//--- все готово, индикатор создан и значения рассчитаны
    return true;
}

//+-----+
//| Ищет два последних перелома зигзага и помещает в массивы |
//+-----+

bool GetLastTwoFractures(double &get_values[],datetime &get_times[],int handle)
{
    double values[];           // массив для получения значений зигзага
    datetime times[];          // массив для получения времени
    int size=100;               // размер массивов
}

```

```

ResetLastError();

//--- копируем 100 последних значений индикатора
int copied=CopyBuffer(handle,0,0,size,values);
//--- проверим сколько скопировалось
if(copied<100)
{
    PrintFormat("%s: Не удалось скопировать %d значений индикатора с хэндлом=%d. Код ошибки=%d", __FUNCTION__,size,handle,GetLastError());
    return false;
}

//--- определим порядок доступа к массиву как в таймсерии
ArraySetAsSeries(values,true);

//--- сюда запишем номера баров, на которых найдены переломы
int positions[];

//--- зададим размеры массивов
ArrayResize(get_values,3); ArrayResize(get_times,3); ArrayResize(positions,3);
//--- счетчики
int i=0,k=0;
//--- начинаем поиск переломов
while(i<100)
{
    double v=values[i];
    //--- пустые значения нас не интересуют
    if(v!=0.0)
    {
        //--- запомним номер бара
        positions[k]=i;
        //--- запомним значение зигзага на переломе
        get_values[k]=values[i];
        PrintFormat("%s: Zigzag[%d]=%G", __FUNCTION__,i,values[i]);
        //--- увеличим счетчик
        k++;
        //--- если нашли два перелома, то ломаем цикл
        if(k>2) break;
    }
    i++;
}
//--- определим порядок доступа к массивам как в таймсерии
ArraySetAsSeries(times,true); ArraySetAsSeries(get_times,true);
if(CopyTime(_Symbol,_Period,0,size,times)<=0)
{
    PrintFormat("%s: Не удалось скопировать %d значений из CopyTime(). Код ошибки=%d", __FUNCTION__,size,GetLastError());
    return false;
}

//--- найдем время открытия баров, на которых были 2 последних перелома
get_times[0]=times[positions[1]]; // предпоследнее значение будет записано первым переломом
get_times[1]=times[positions[2]]; // третье с конца значение будет вторым переломом
PrintFormat("%s: первый=%s, второй=%s", __FUNCTION__,TimeToString(get_times[1]),TimeToString(get_times[2]));

```

```
//--- все получилось
return true;
}
```

Смотри также

[ChartApplyTemplate\(\)](#), [Ресурсы](#)

## ChartWindowFind

Возвращает номер подокна, в котором находится индикатор. Существует 2 варианта функции.

1. Функция ищет на указанном графике подокно с указанным "коротким именем" индикатора (короткое имя выводится слева вверху подокна) и в случае удачи возвращает номер подокна.

```
int ChartWindowFind(
    long    chart_id,           // идентификатор графика
    string  indicator_shortname // короткое имя индикатора, см INDICATOR\_SHORTNAME
```

2. Функция должна вызываться из пользовательского индикатора и возвращает номер подокна, в котором этот индикатор работает.

```
int ChartWindowFind();
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*indicator\_shortname*

[in] Короткое имя индикатора.

### Возвращаемое значение

Номер подокна в случае удачи. Ноль означает главное окно графика. В случае неудачи возвращает -1.

### Примечание

Если вызывается второй вариант функции (без параметров) из скрипта или эксперта, то она возвращает -1.

Не следует путать короткое имя индикатора и имя файла, которое указывается при создании индикатора функциями [iCustom\(\)](#) и [IndicatorCreate\(\)](#). Если короткое наименование индикатора не задается явным образом, то при компиляции в нем указывается имя файла с исходным кодом индикатора.

Необходимо правильно формировать короткое имя индикатора, которое с помощью функции [IndicatorSetString\(\)](#) записывается в свойство [INDICATOR\\_SHORTNAME](#). Мы рекомендуем, чтобы короткое имя содержало значения входных параметров индикатора, так как идентификация удаляемого с графика индикатора в функции [ChartIndicatorDelete\(\)](#) производится именно по короткому имени.

### Пример:

```
#property script_show_inputs
//--- input parameters
input string  shortName="MACD(12,26,9)";
//-----+
//| Сообщает номер окна графика с указанным индикатором |
//+-----+
int GetIndicatorSubWindowNumber (long chartID=0,string short_name="" )
```

```
{  
    int window=-1;  
//---  
    if((ENUM_PROGRAM_TYPE)MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR)  
    {  
        //--- функция вызвана из индикатора, имя не требуется  
        window=ChartWindowFind();  
    }  
    else  
    {  
        //--- функция вызвана из эксперта или скрипта  
        window=ChartWindowFind(0,short_name);  
        if(window== -1) Print(__FUNCTION__+"(): Error = ",GetLastError());  
    }  
//---  
    return (window);  
}  
//-----+  
//| Script program start function |  
//-----+  
void OnStart()  
{  
//---  
    int window=GetIndicatorSubWindowNumber(0,shortName);  
    if(window!= -1)  
        Print("Индикатор "+shortName+" находится в окне №"+(string)window);  
    else  
        Print("Индикатор "+shortName+" не найден. window = "+(string)window);  
}
```

Смотри также

[ObjectCreate\(\)](#), [ObjectFind\(\)](#)

## ChartTimePriceToXY

Преобразует координаты графика из представления времени/цена в координаты по осям X и Y.

```
bool ChartTimePriceToXY(
    long      chart_id,        // идентификатор графика
    int       sub_window,      // номер подокна
    datetime  time,           // время на графике
    double    price,           // цена на графике
    int&     x,               // координата X для времени на графике
    int&     y                // координата Y для цены на графике
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*sub\_window*

[in] Номер подокна графика. 0 означает главное окно графика.

*time*

[in] Значение времени на графике, для которого будет получено значение в пикселях на оси X. Начало координат находится в левом верхнем углу главного окна графика.

*price*

[in] Значение цены на графике, для которого будет получено значение в пикселях на оси Y. Начало координат находится в левом верхнем углу главного окна графика.

*x*

[out] Переменная, в которую будет получено преобразование времени в координату X.

*y*

[out] Переменная, в которую будет получено преобразование цены в координату Y.

### Возвращаемое значение

Возвращает true в случае успешного выполнения, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Смотри также

[ChartXYToTimePrice\(\)](#)

## ChartXYToTimePrice

Преобразует координаты X и Y графика в значения время и цена.

```
bool ChartXYToTimePrice(
    long      chart_id,      // идентификатор графика
    int       x,             // координата X на графике
    int       y,             // координата Y на графике
    int&     sub_window,    // номер подокна
    datetime& time,         // время на графике
    double&   price,        // цена на графике
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*x*

[in] Координата X.

*y*

[in] Координата Y.

*sub\_window*

[out] Переменная, в которую будет записан номер подокна графика. 0 означает главное окно графика.

*time*

[out] Значение времени на графике, для которого будет получено значение в пикселях на оси X. Начало координат находится в левом верхнем углу главного окна графика.

*price*

[out] Значение цены на графике, для которого будет получено значение в пикселях на оси Y. Начало координат находится в левом верхнем углу главного окна графика.

### Возвращаемое значение

Возвращает true в случае успешного выполнения, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Пример:

```
//+-----+
//| ChartEvent function
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
    //--- выведем параметры события на график
```

```

Comment(__FUNCTION__," : id=",id," lparam=",lparam," dparam=",dparam," sparam=",sparam)
//--- если это события клика мышки на графике
if(id==CHARTEVENT_CLICK)
{
    //--- подготовим переменные
    int      x      =(int)lparam;
    int      y      =(int)dparam;
    datetime dt     =0;
    double   price =0;
    int      window=0;
//--- преобразуем координаты X и Y в терминах дата/время
if(ChartXYToTimePrice(0,x,y,window,dt,price))
{
    PrintFormat("Window=%d X=%d Y=%d => Time=%s Price=%G",window,x,y,TimeToString(dt),price);
    //--- сделаем обратное преобразование: (X,Y) => (Time,Price)
    if(ChartTimePriceToXY(0,window,dt,price,x,y))
        PrintFormat("Time=%s Price=%G => X=%d Y=%d",TimeToString(dt),price,x,y);
    else
        Print("ChartTimePriceToXY return error code: ",GetLastError());
    //--- удаляем линии
    ObjectDelete(0,"V Line");
    ObjectDelete(0,"H Line");
    //--- создаем горизонтальную и вертикальную линии перекрестия
    ObjectCreate(0,"H Line",OBJ_HLINE,window,dt,price);
    ObjectCreate(0,"V Line",OBJ_VLINE,window,dt,price);
    ChartRedraw(0);
}
else
{
    Print("ChartXYToTimePrice return error code: ",GetLastError());
    Print("-----+");
}
}

```

## Смотри также

## ChartTimePriceToXY()

## ChartOpen

Открывает новый график с указанным символом и периодом.

```
long ChartOpen(
    string          symbol,      // имя символа
    ENUM_TIMEFRAMES period      // период
);
```

### Параметры

*symbol*

[in] Символ графика. [NULL](#) означает символ текущего графика (к которому прикреплен эксперт).

*period*

[in] Период графика (таймфрейм). Может принимать одно из значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

### Возвращаемое значение

При успешном открытии графика функция возвращает идентификатор графика. В противном случае возвращает 0.

### Примечание

Максимально возможное количество одновременно открытых графиков в терминале не может быть больше, чем значение [CHARTS\\_MAX](#).

## ChartFirst

Возвращает идентификатор первого графика клиентского терминала.

```
long ChartFirst();
```

### Возвращаемое значение

Идентификатор графика.

## ChartNext

Функция возвращает идентификатор графика, следующего за указанным.

```
long ChartNext(
    long chart_id          // идентификатор графика
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 не означает текущий график. 0 означает "вернуть идентификатор первого графика".

### Возвращаемое значение

Идентификатор графика. Если список графиков закончился, функция возвращает -1.

### Пример:

```
//--- переменные для идентификаторов графиков
long currChart,prevChart=ChartFirst();
int i=0,limit=100;
Print("ChartFirst = ",ChartSymbol(prevChart)," ID = ",prevChart);
while(i<limit)// у нас наверняка не больше 100 открытых графиков
{
    currChart=ChartNext(prevChart); // на основании предыдущего получим новый график
    if(currChart<0) break;           // достигли конца списка графиков
    Print(i,ChartSymbol(currChart)," ID =",currChart);
    prevChart=currChart;// запомним идентификатор текущего графика для ChartNext()
    i++; // не забудем увеличить счетчик
}
```

## ChartClose

Закрывает указанный график.

```
bool ChartClose(
    long chart_id=0           // идентификатор графика
);
```

### Параметры

*chart\_id=0*

[in] Идентификатор графика. 0 означает текущий график.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## ChartSymbol

Возвращает имя символа указанного графика.

```
string ChartSymbol(
    long chart_id=0           // идентификатор графика
);
```

### Параметры

*chart\_id=0*

[in] Идентификатор графика. 0 означает текущий график.

### Возвращаемое значение

Если графика не существует, то возвращается пустая строка.

### Смотри также

[ChartSetSymbolPeriod](#)

## ChartPeriod

Возвращает значение [периода](#) указанного графика.

```
ENUM_TIMEFRAMES ChartPeriod(
    long chart_id=0           // идентификатор графика
);
```

### Параметры

*chart\_id=0*

[in] Идентификатор графика. 0 означает текущий график.

### Возвращаемое значение

Значение типа [ENUM\\_TIMEFRAMES](#). Если графика не существует, то возвращается 0.

## ChartRedraw

Вызывает принудительную перерисовку указанного графика.

```
void ChartRedraw(
    long chart_id=0           // идентификатор графика
);
```

### Параметры

*chart\_id=0*

[in] Идентификатор графика. 0 означает текущий график.

### Примечание

Обычно применяется после изменения [свойств объектов](#).

### Смотри также

[Графические объекты](#)

## ChartSetDouble

Задает значение соответствующего свойства указанного графика. Свойство графика должно быть типа [double](#). Отданная команда поступает в очередь сообщений графика и выполняется только после обработки всех предыдущих команд.

```
bool ChartSetDouble(
    long   chart_id,      // идентификатор графика
    int    prop_id,       // идентификатор свойства
    double value         // значение
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*prop\_id*

[in] Идентификатор свойства графика. Значение может быть одним из значений перечисления [ENUM\\_CHART\\_PROPERTY\\_DOUBLE](#) (кроме read-only свойств).

*value*

[in] Значение свойства.

### Возвращаемое значение

Возвращает true в случае удачного помещения команды в очередь графика, иначе false. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция является асинхронной - это означает, что функция не дожидается выполнения команды, успешно поставленной в очередь указанного графика, а сразу же возвращает управление. Изменение свойства произойдет только после обработки команды в очереди графика. Для немедленного выполнения команд в очереди графика нужно вызвать функцию [ChartRedraw](#).

Если требуется немедленно изменить сразу несколько свойств графика, то необходимо использовать соответствующие функции ([ChartSetString](#), [ChartSetDouble](#), [ChartSetString](#)) выполнить в одном блоке кода и затем вызвать один раз [ChartRedraw](#).

Для проверки результата выполнения можно использовать функцию, запрашивающую указанное свойство графика ([ChartGetInteger](#), [ChartGetDouble](#), [ChartGetString](#)). При этом необходимо иметь в виду, что данные функции являются синхронными и дожидаются результата выполнения.

## ChartSetInteger

Задает значение соответствующего свойства указанного графика. Свойство графика должно быть типов [datetime](#), [int](#), [color](#), [bool](#) или [char](#). Отданная команда поступает в очередь сообщений графика и выполняется только после обработки всех предыдущих команд.

```
bool ChartSetInteger(
    long chart_id,           // идентификатор графика
    int prop_id,             // идентификатор свойства
    long value               // значение
);
```

Задает значение соответствующего свойства указанного подокна.

```
bool ChartSetInteger(
    long chart_id,           // идентификатор графика
    int prop_id,             // идентификатор свойства
    int sub_window,          // номер подокна
    long value               // значение
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*prop\_id*

[in] Идентификатор свойства графика. Значение может быть одним из значений перечисления [ENUM\\_CHART\\_PROPERTY\\_INTEGER](#) (кроме read-only свойств).

*sub\_window*

[in] Номер подокна графика. Для первого варианта по умолчанию значение равно 0 (главное окно графика). Большинство свойств не требуют указания номера подокна.

*value*

[in] Значение свойства.

### Возвращаемое значение

Возвращает true в случае удачного помещения команды в очередь графика, иначе false. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция является асинхронной - это означает, что функция не дожидается выполнения команды, успешно поставленной в очередь указанного графика, а сразу же возвращает управление. Изменение свойства произойдет только после обработки команды в очереди графика. Для немедленного выполнения команд в очереди графика нужно вызвать функцию [ChartRedraw](#).

Если требуется немедленно изменить сразу несколько свойств графика, то необходимо соответствующие функции ([ChartSetString](#), [ChartSetDouble](#), [ChartSetString](#)) выполнить в одном блоке кода и затем вызвать один раз [ChartRedraw](#).

Для проверки результата выполнения можно использовать функцию, запрашивающую указанное свойство графика ([ChartGetInteger](#), [ChartGetDouble](#), [ChartSetString](#)). При этом необходимо иметь в виду, что данные функции являются синхронными и дожидаются результата выполнения.

### Пример:

```

//+-----+
//| Expert initialization function
//+-----+
void OnInit()
{
    //--- включение сообщений о перемещении мыши по окну чарта
    ChartSetInteger(0,CHART_EVENT_MOUSE_MOVE,1);
    //--- принудительное обновление свойств графика гарантирует готовность к обработке сообщений
    ChartRedraw();
}
//+-----+
//| MouseState
//+-----+
string MouseState(uint state)
{
    string res;
    res+="\nML: " +(((state& 1)== 1)? "DN": "UP"); // mouse left
    res+="\nMR: " +(((state& 2)== 2)? "DN": "UP"); // mouse right
    res+="\nMM: " +(((state&16)==16)? "DN": "UP"); // mouse middle
    res+="\nMX: " +(((state&32)==32)? "DN": "UP"); // mouse first X key
    res+="\nMY: " +(((state&64)==64)? "DN": "UP"); // mouse second X key
    res+="\nSHIFT: "+(((state& 4)== 4)? "DN": "UP"); // shift key
    res+="\nCTRL: " +(((state& 8)== 8)? "DN": "UP"); // control key
    return(res);
}
//+-----+
//| ChartEvent function
//+-----+
void OnChartEvent(const int id,const long &lparam,const double &dparam,const string &sparam)
{
    if(id==CHARTEVENT_MOUSE_MOVE)
        Comment("POINT: ",(int)lparam,",",(int)dparam,"\\n",MouseState((uint)sparam));
}

```

## ChartSetString

Задает значение соответствующего свойства указанного графика. Свойство графика должно быть типа `string`. Отданная команда поступает в очередь сообщений графика и выполняется только после обработки всех предыдущих команд.

```
bool ChartSetString(
    long chart_id,           // идентификатор графика
    int prop_id,             // идентификатор свойства
    string str_value         // значение
);
```

### Параметры

`chart_id`

[in] Идентификатор графика. 0 означает текущий график.

`prop_id`

[in] Идентификатор свойства графика. Значение может быть одним из значений перечисления [ENUM\\_CHART\\_PROPERTY\\_STRING](#) (кроме `read-only` свойств).

`str_value`

[in] Стока для установки свойства. Длина строки не может превышать 2045 символов (лишние символы будут обрезаны).

### Возвращаемое значение

Возвращает `true` в случае удачного помещения команды в очередь графика, иначе `false`. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция является асинхронной - это означает, что функция не дожидается выполнения команды, успешно поставленной в очередь указанного графика, а сразу же возвращает управление. Изменение свойства произойдет только после обработки команды в очереди графика. Для немедленного выполнения команд в очереди графика нужно вызвать функцию [ChartRedraw](#).

Если требуется немедленно изменить сразу несколько свойств графика, то необходимо использовать соответствующие функции ([ChartSetString](#), [ChartSetDouble](#), [ChartSetString](#)) выполнить в одном блоке кода и затем вызвать один раз [ChartRedraw](#).

Для проверки результата выполнения можно использовать функцию, запрашивающую указанное свойство графика ([ChartGetInteger](#), [ChartGetDouble](#), [ChartGetString](#)). При этом необходимо иметь в виду, что данные функции являются синхронными и дожидаются результата выполнения.

### Пример:

```
void OnTick()
{
//---
    double Ask,Bid;
    int Spread;
    Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
```

```
Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
string comment=StringFormat("Выводим цены:\nAsk = %G\nBid = %G\nSpread = %d",
                           Ask,Bid,Spread);
ChartSetString(0,CHART_COMMENT,comment);
}
```

Смотри также

[Comment](#), [ChartGetString](#)

## ChartGetDouble

Возвращает значение соответствующего свойства указанного графика. Свойство графика должно быть типа double. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
double ChartGetDouble(
    long chart_id,           // идентификатор графика
    int prop_id,             // идентификатор свойства
    int sub_window=0          // номер подокна, если требуется
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool ChartGetDouble(
    long chart_id,           // идентификатор графика
    int prop_id,             // идентификатор свойства
    int sub_window,           // номер подокна
    double& double_var        // сюда примем значение свойства
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*prop\_id*

[in] Идентификатор свойства графика. Значение может быть одним из значений перечисления [ENUM\\_CHART\\_PROPERTY\\_DOUBLE](#).

*sub\_window*

[in] Номер подокна графика. Для первого варианта по умолчанию значение равно 0 (главное окно графика). Большинство свойств не требуют указания номера подокна.

*double\_var*

[out] Переменная типа double, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа double.

Для второго варианта вызова возвращает true, если данное свойство поддерживается и значение было помещено в переменную *double\_var*, иначе возвращает false. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция является синхронной - это означает, что она дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом.

### Пример:

```
void OnStart()
{
    double priceMin=ChartGetDouble(0,CHART_PRICE_MIN,0);
    double priceMax=ChartGetDouble(0,CHART_PRICE_MAX,0);
    Print("CHART_PRICE_MIN = ",priceMin);
    Print("CHART_PRICE_MAX = ",priceMax);
}
```

## ChartGetInteger

Возвращает значение соответствующего свойства указанного графика. Свойство графика должно быть типов [datetime](#), [int](#) или [bool](#). Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
long ChartGetInteger(
    long chart_id,           // идентификатор графика
    int prop_id,             // идентификатор свойства
    int sub_window=0          // номер подокна, если требуется
);
```

2. Возвращает `true` или `false` в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool ChartGetInteger(
    long chart_id,           // идентификатор графика
    int prop_id,             // идентификатор свойства
    int sub_window,           // номер подокна
    long& long_var            // сюда примем значение свойства
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*prop\_id*

[in] Идентификатор свойства графика. Значение может быть одним из значений перечисления [ENUM\\_CHART\\_PROPERTY\\_INTEGER](#).

*sub\_window*

[in] Номер подокна графика. Для первого варианта по умолчанию значение равно 0 (главное окно графика). Большинство свойств не требуют указания номера подокна.

*long\_var*

[out] Переменная типа `long`, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа `long`.

Для второго варианта вызова возвращает `true`, если данное свойство поддерживается и значение было помещено в переменную `long_var`, иначе возвращает `false`. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция является синхронной - это означает, что она дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом.

### Пример:

```
void OnStart()
{
    int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);
    int width=ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0);
    Print("CHART_HEIGHT_IN_PIXELS = ",height," pixels");
    Print("CHART_WIDTH_IN_PIXELS = ",width," pixels");
}
```

## ChartGetString

Возвращает значение соответствующего свойства указанного графика. Свойство графика должно быть типа `string`. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
string ChartGetString(
    long chart_id,           // идентификатор графика
    int prop_id              // идентификатор свойства
);
```

2. Возвращает `true` или `false` в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool ChartGetString(
    long chart_id,           // идентификатор графика
    int prop_id,             // идентификатор свойства
    string& string_var      // сюда примем значение свойства
);
```

### Параметры

`chart_id`

[in] Идентификатор графика. 0 означает текущий график.

`prop_id`

[in] Идентификатор свойства графика. Значение может быть одним из значений перечисления [ENUM\\_CHART\\_PROPERTY\\_STRING](#).

`string_var`

[out] Переменная типа `string`, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа `string`.

Для второго варианта вызова возвращает `true`, если данное свойство поддерживается и значение было помещено в переменную `string_var`, иначе возвращает `false`. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция `ChartGetString` может использоваться для считывания комментариев, выведенных на график функциями [Comment](#) или [ChartSetString](#).

Функция является синхронной - это означает, что она дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом.

### Пример:

```
void OnStart()
{
```

```
ChartSetString(0,CHART_COMMENT,"Test comment.\nSecond line.\nThird!");  
ChartRedraw();  
Sleep(1000);  
string comm=ChartGetString(0,CHART_COMMENT);  
Print(comm);  
}
```

Смотри также

[Comment](#), [ChartSetString](#)

## ChartNavigate

Осуществляет сдвиг указанного графика на указанное количество баров относительно указанной позиции графика.

```
bool ChartNavigate(
    long          chart_id,      // идентификатор графика
    ENUM_CHART_POSITION position, // позиция
    int           shift=0        // значение сдвига
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*position*

[in] Позиция графика, относительно которой будет произведено смещение. Значение может быть одним из значений перечисления [ENUM\\_CHART\\_POSITION](#).

*shift=0*

[in] Количество баров, на которое необходимо сместить график. Положительное значение означает смещение вправо (к концу графика), отрицательное значение означает смещение влево (к началу графика). Нулевое смещение имеет смысл, когда производится навигация к началу или концу графика.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- получим handle текущего графика
long handle=ChartID();
string comm="";
if(handle>0) // если получилось, дополнительно настроим
{
//--- отключим автопрокрутку
ChartSetInteger(handle,CHART_AUTOSCROLL,false);
//--- установим отступ правого края графика
ChartSetInteger(handle,CHART_SHIFT,true);
//--- отобразим в виде свечей
ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
//--- установить режим отображения тиковых объемов
ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);

//--- подготовим текст для вывода в Comment()
}
```

```

comm="Прокрутим на 10 баров вправо от начала истории";
//--- выводем комментарий
Comment(comm);
//--- прокрутим на 10 баров вправо от начала истории
ChartNavigate(handle,CHART_BEGIN,10);
//--- получим номер самого первого видимого на графике бара (нумерация как в таймфрейме)
long first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
//--- добавим символ переноса строки
comm=comm+"\r\n";
//--- дополним коментарий
comm=comm+"Первый бар на графике имеет номер "+IntegerToString(first_bar)+"\r\n";
//--- выводем комментарий
Comment(comm);
//--- подождем 5 секунд, чтобы успеть увидеть, как двигается график
Sleep(5000);

//--- допишем текст комментария
comm=comm+"\r\n"+"Прокрутим на 10 баров влево от правого края графика";
Comment(comm);
//--- прокрутим на 10 баров влево от правого края графика
ChartNavigate(handle,CHART_END,-10);
//--- получим номер самого первого видимого на графике бара (нумерация как в таймфрейме)
first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
comm=comm+"\r\n";
comm=comm+"Первый бар на графике имеет номер "+IntegerToString(first_bar)+"\r\n";
Comment(comm);
//--- подождем 5 секунд, чтобы успеть увидеть, как двигается график
Sleep(5000);

//--- новый блок прокрутки графика
comm=comm+"\r\n"+"Прокрутим на 300 баров вправо от начала истории";
Comment(comm);
//--- прокрутим на 300 баров вправо от начала истории
ChartNavigate(handle,CHART_BEGIN,300);
first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
comm=comm+"\r\n";
comm=comm+"Первый бар на графике имеет номер "+IntegerToString(first_bar)+"\r\n";
Comment(comm);
//--- подождем 5 секунд, чтобы успеть увидеть, как двигается график
Sleep(5000);

//--- новый блок прокрутки графика
comm=comm+"\r\n"+"Прокрутим на 300 баров влево от правого края графика";
Comment(comm);
//--- прокрутим на 300 баров влево от правого края графика
ChartNavigate(handle,CHART_END,-300);
first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
comm=comm+"\r\n";
comm=comm+"Первый бар на графике имеет номер "+IntegerToString(first_bar)+"\r\n";

```

```
Comment (comm) ;  
}  
}
```

## ChartID

Возвращает идентификатор текущего графика.

```
long ChartID();
```

### Возвращаемое значение

Значение типа [long](#).

## ChartIndicatorAdd

Добавляет на указанное окно графика индикатор с указанным хэндлом. Индикатор и график должны быть построены на одинаковых символе и таймфрейме.

```
bool ChartIndicatorAdd(
    long chart_id,           // идентификатор графика
    int sub_window,          // номер подокна
    int indicator_handle     // хэндл индикатора
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*sub\_window*

[in] Номер подокна графика. 0 означает главное окно графика. Чтобы добавить индикатор в новое окно, параметр должен быть на единицу больше, чем индекс последнего существующего окна, то есть равен [CHART\\_WINDOWS\\_TOTAL](#). Если значение параметра превышает значение [CHART\\_WINDOWS\\_TOTAL](#), то новое окно создано не будет, индикатор не будет добавлен.

*indicator\_handle*

[in] Хэндл индикатора.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#). Ошибка 4114 означает, что график и добавляемый индикатор отличаются по символу или таймфрейму.

### Примечание

Если на главное окно графика добавляется индикатор, который должен быть отрисован в отдельном подокне (например, встроенный [iMACD](#) или пользовательский индикатор с указанным свойством [#property indicator\\_separate\\_window](#)), то такой индикатор может оказаться невидимым, хотя и будет присутствовать в списке индикаторов. Это означает, что масштаб данного индикатора отличается от масштаба графика цен и значения добавленного индикатора не попадают в отображаемый диапазон ценового графика. При этом [GetLastError\(\)](#) вернет нулевой код, означающий отсутствие ошибки. Значения такого "невидимого" индикатора можно будет наблюдать в "Окне данных" и получать из других MQL5-программ.

### Пример:

```
#property description "Эксперт для демонстрации работы с функцией ChartIndicatorAdd().
#property description "После запуска на графике (и получения ошибки в Журнал) откройте
#property description "свойства советника и задайте правильные параметры <symbol> и <per
#property description "На график будет добавлен индикатор MACD.

//--- input parameters
input string          symbol="AUDUSD";      // имя символа
input ENUM_TIMEFRAMES period=PERIOD_M12; // таймфрейм
```

```

input int      fast_ema_period=12;           // период быстрой средней MACD
input int      slow_ema_period=26;           // период медленной средней MACD
input int      signal_period=9;              // период усреднения разности
input ENUM_APPLIED_PRICE apr=PRICE_CLOSE;   // тип цены для расчета MACD

int indicator_handle=INVALID_HANDLE;
//+-----+
//| Expert initialization function          |
//+-----+
int OnInit()
{
//---
    indicator_handle=iMACD(symbol,period,fast_ema_period,slow_ema_period,signal_period,
//--- попробуем добавить индикатор на график
    if(!AddIndicator())
    {
//--- функция AddIndicator() отказалась добавить индикатор на график
        int answer=MessageBox("Всё равно попытаться добавить MACD на график?",
                            "Неправильно задан символ и/или таймфрейм для добавления индикатора",
                            MB_YESNO // будут показаны кнопки выбора "Yes" и "No"
                           );
//--- если пользователь все равно настаивает на неправильном использовании Chart
        if(answer==IDYES)
        {
//--- сначала сообщим об этом в Журнал
            PrintFormat("Внимание! %s: Попробуем добавить индикатор MACD(%s/%s) на график",
                        _FUNCTION_,symbol,EnumToString(period),_Symbol,EnumToString(_Period));
//--- получим номер нового подокна, в которое попытаемся добавить индикатор
            int subwindow=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
//--- теперь сделаем попытку, обреченную на ошибку
            if(!ChartIndicatorAdd(0,subwindow,indicator_handle))
                PrintFormat("Не удалось добавить индикатор MACD на окно %d графика. Код ошибки: %d",
                            subwindow,GetLastError());
        }
    }
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert tick function                    |
//+-----+
void OnTick()
{
// эксперт ничего не делает
}
//+-----+
//| Функция проверки и добавления индикатора на график                         |
//+-----+
bool AddIndicator()

```

```

{
//--- выводимое сообщение
string message;
//--- проверим на совпадение символ индикатора и символ графика
if(symbol!=_Symbol)
{
    message="Демонстрация использования функции Demo_ChartIndicatorAdd() :";
    message=message+"\r\n";
    message=message+"Нельзя на график добавить индикатор, рассчитанный на другом си";
    message=message+"\r\n";
    message=message+"Укажите в свойствах эксперта символ графика - "+_Symbol+(".");
    Alert(message);
//--- досрочный выход, не будем добавлять индикатор на график
return false;
}
//--- проверим на совпадение таймфрейм индикатора и таймфрейм графика
if(period!=_Period)
{
    message="Нельзя на график добавить индикатор, рассчитанный на другом таймфрейме.";
    message=message+"\r\n";
    message=message+"Укажите в свойствах эксперта таймфрейм графика - "+EnumToString(_Period);
    Alert(message);
//--- досрочный выход, не будем добавлять индикатор на график
return false;
}
//--- все проверки прошли, символ и период индикатора соответствуют графику
if(indicator_handle==INVALID_HANDLE)
{
    Print(__FUNCTION__," Создаем индикатор MACD");
    indicator_handle=iMACD(symbol,period,fast_ema_period,slow_ema_period,signal_peri;
    if(indicator_handle==INVALID_HANDLE)
    {
        Print("Не удалось создать индикатор MACD. Код ошибки ",GetLastError());
    }
}
//--- сбросим код ошибки
ResetLastError();
//--- накладываем индикатор на график
Print(__FUNCTION__," Добавляем индикатор MACD на график");
Print("MACD построен на ",symbol,"/",EnumToString(period));
//--- получим номер нового подокна, в которое добавим индикатор MACD
int subwindow=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
PrintFormat("Добавляем индикатор MACD на окно %d графика",subwindow);
if(!ChartIndicatorAdd(0,subwindow,indicator_handle))
{
    PrintFormat("Не удалось добавить индикатор MACD на окно %d графика. Код ошибки
                subwindow,GetLastError());
}
//--- добавление индикатора на график прошло успешно

```

```
    return(true);
}
```

**Смотри также**

[ChartIndicatorDelete\(\)](#), [ChartIndicatorName\(\)](#), [ChartIndicatorsTotal\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#)

## ChartIndicatorDelete

Удаляет с указанного окна графика индикатор с указанным именем.

```
bool ChartIndicatorDelete(
    long      chart_id,           // идентификатор графика
    int       sub_window,         // номер подокна
    const string indicator_shortname // короткое имя индикатора
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*sub\_window*

[in] Номер подокна графика. 0 означает главное окно графика.

*const indicator\_shortname*

[in] Короткое имя индикатора, которое задается в свойстве [INDICATOR\\_SHORTNAME](#) функцией [IndicatorSetString\(\)](#). Получить короткое имя индикатора можно функцией [ChartIndicatorName\(\)](#).

### Возвращаемое значение

Возвращает true в случае успешного удаления индикатора, иначе false. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Если в указанном подокне графика существует несколько индикаторов с одинаковым коротким именем, то будет удален первый по порядку.

Если на значениях удаляемого индикатора построены другие индикаторы на этом же графике, то они также будут удалены.

Не следует путать короткое имя индикатора и имя файла, которое указывается при создании индикатора функциями [iCustom\(\)](#) и [IndicatorCreate\(\)](#). Если короткое наименование индикатора не задается явным образом, то при компиляции в нем указывается имя файла, содержащего исходный код индикатора.

Удаление индикатора с графика не означает, что расчетная часть индикатора также будет удалена из памяти терминала. Для освобождения хэндла индикатора используйте функцию [IndicatorRelease\(\)](#).

Необходимо правильно формировать короткое имя индикатора, которое с помощью функции [IndicatorSetString\(\)](#) записывается в свойство [INDICATOR\\_SHORTNAME](#). Мы рекомендуем, чтобы короткое имя содержало значения входных параметров индикатора, так как идентификация удаляемого с графика индикатора в функции [ChartIndicatorDelete\(\)](#) производится именно по короткому имени.

### Пример удаления индикатора при неудачной инициализации:

```
//+-----+
//|                               Demo_ChartIndicatorDelete.mq5 |
//| Copyright 2011, MetaQuotes Software Corp. |
//|
```

```
//| https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Histogram
#property indicator_label1 "Histogram"
#property indicator_type1  DRAW_HISTOGRAM
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameters
input int    first_param=1;
input int    second_param=2;
input int    third_param=3;
input bool   wrong_init=true;
//--- indicator buffers
double      HistogramBuffer[];
string      shortname;
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
    int res=INIT_SUCCEEDED;
//--- привяжем массив HistogramBuffer к индикаторному буферу
    SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- сконструируем короткое имя индикатора на основе входных параметров
    shortname=StringFormat("Demo_ChartIndicatorDelete(%d,%d,%d)",
                           first_param,second_param,third_param);
    IndicatorSetString(INDICATOR_SHORTNAME,shortname);
//--- если задано принудительное завершение индикатора, вернем ненулевое значение
    if(wrong_init) res=INIT_FAILED;
    return(res);
}
//+-----+
//| Custom indicator iteration function               |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
```

```

        const long &volume[],
        const int &spread[])
    {
//--- начальная позиция для работы в цикле
    int start=prev_calculated-1;
    if(start<0) start=0;
//--- заполняем индикаторный буфер значениями
    for(int i=start;i<rates_total;i++)
    {
        HistogramBuffer[i]=close[i];
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}
//----------------------------------------------------------------------------------------------------------------+
//| обработчик события Deinit
//----------------------------------------------------------------------------------------------------------------+
void OnDeinit(const int reason)
{
    PrintFormat("%s: Код причины deinициализации=%d", __FUNCTION__, reason);
    if(reason==REASON_INITFAILED)
    {
        PrintFormat("Индикатор с коротким именем %s (файл %s) удаляет себя с графика",shortname, __FILE__);
        int window=ChartWindowFind();
        bool res=ChartIndicatorDelete(0,window,shortname);
//--- проанализируем результат вызова ChartIndicatorDelete()
        if(!res)
        {
            PrintFormat("Не удалось удалить индикатор %s с окна #%d. Код ошибки %d",
                       shortname,window,GetLastError());
        }
    }
}
}

```

#### Смотри также

[ChartIndicatorAdd\(\)](#), [ChartIndicatorName\(\)](#), [ChartIndicatorsTotal\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#)

## ChartIndicatorGet

Возвращает хэндл индикатора с указанным коротким именем на указанном окне графика.

```
int ChartIndicatorGet(
    long      chart_id,           // идентификатор графика
    int       sub_window,         // номер подокна
    const string indicator_shortname // короткое имя индикатора
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*sub\_window*

[in] Номер подокна графика. 0 означает главное окно графика.

*const indicator\_shortname*

[in] Короткое имя индикатора, которое задается в свойстве [INDICATOR\\_SHORTNAME](#) функцией [IndicatorSetString\(\)](#). Получить короткое имя индикатора можно функцией [ChartIndicatorName\(\)](#).

### Возвращаемое значение

Возвращает хэндл индикатора в случае успешного выполнения, иначе [INVALID\\_HANDLE](#). Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Необходимо правильно формировать короткое имя индикатора при его создании, которое с помощью функции [IndicatorSetString\(\)](#) записывается в свойство [INDICATOR\\_SHORTNAME](#). Рекомендуется, чтобы короткое имя содержало значения входных параметров индикатора, так как идентификация индикатора в функции [ChartIndicatorGet\(\)](#) производится именно по короткому имени.

Другой способ идентификации индикатора - получить список его параметров по заданному хэндлу с помощью функции [IndicatorParameters\(\)](#) и затем провести анализ полученных значений.

### Пример:

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    //--- количество окон на графике (всегда есть хотя бы одно главное окно)
    int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
    //--- проходим по окнам
    for(int w=0;w<windows;w++)
    {
        //--- сколько индикаторов в данном окне/подокне
        int total=ChartIndicatorsTotal(0,w);
        //--- переберем все индикаторы в окне
```

```
for(int i=0;i<total;i++)
{
    //--- получим короткое имя индикатора
    string name=ChartIndicatorName(0,w,i);
    //--- получим хэндл индикатора
    int handle=ChartIndicatorGet(0,w,name);
    //--- выведем в журнал
    PrintFormat("Window=%d, index=%d, Name=%s, handle=%d",w,i,name,handle);
    //--- обязательно освобождаем хендл индикатора, как только он становится ненужным
    IndicatorRelease(handle);
}
}
```

#### Смотри также

[ChartIndicatorAdd\(\)](#), [ChartIndicatorName\(\)](#), [ChartIndicatorsTotal\(\)](#), [IndicatorParameters\(\)](#)

## ChartIndicatorName

Возвращает короткое имя индикатора по номеру в списке индикаторов на указанном окне графика.

```
string ChartIndicatorName(
    long chart_id,           // идентификатор графика
    int sub_window,          // номер подокна
    int index                // индекс индикатора в списке индикаторов, добавленных к данно
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*sub\_window*

[in] Номер подокна графика. 0 означает главное окно графика.

*index*

[in] Индекс индикатора с списке индикаторов. Нумерация индикаторов начинается с нуля, то есть самый первый индикатор в списке имеет нулевой индекс. Количество индикаторов в списке можно получить функцией [ChartIndicatorsTotal\(\)](#).

### Возвращаемое значение

Короткое имя индикатора, которое задается в свойстве [INDICATOR\\_SHORTNAME](#) функцией [IndicatorSetString\(\)](#). Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Не следует путать короткое имя индикатора и имя файла, которое указывается при создании индикатора функциями [iCustom\(\)](#) и [IndicatorCreate\(\)](#). Если короткое наименование индикатора не задается явным образом, то при компиляции в нем указывается имя файла с исходным кодом индикатора.

Необходимо правильно формировать короткое имя индикатора, которое с помощью функции [IndicatorSetString\(\)](#) записывается в свойство [INDICATOR\\_SHORTNAME](#). Мы рекомендуем, чтобы короткое имя содержало значения входных параметров индикатора, так как идентификация удаляемого с графика индикатора в функции [ChartIndicatorDelete\(\)](#) производится именно по короткому имени.

### Смотри также

[ChartIndicatorAdd\(\)](#), [ChartIndicatorDelete\(\)](#), [ChartIndicatorsTotal\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#)

## ChartIndicatorsTotal

Возвращает количество всех индикаторов, присоединенных к указанному окну графика.

```
int ChartIndicatorsTotal(
    long chart_id,           // идентификатор графика
    int sub_window           // номер подокна
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*sub\_window*

[in] Номер подокна графика. 0 означает главное окно графика.

### Возвращаемое значение

Количество индикаторов на указанном окне графика. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция предназначена для организации перебора всех индикаторов, присоединенных к данному графику. Количество всех окон графика можно получить из свойства [CHART\\_WINDOWS\\_TOTAL](#) функцией [ChartGetInteger\(\)](#).

### Смотри также

[ChartIndicatorAdd\(\)](#), [ChartIndicatorDelete\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#)

## ChartWindowOnDropped

Возвращает номер подокна графика, на которое брошен мышкой данный эксперт или скрипт. 0 означает главное окно графика.

```
int ChartWindowOnDropped();
```

### Возвращаемое значение

Значение типа [int](#).

### Пример:

```
int myWindow=ChartWindowOnDropped();
int windowsTotal=ChartGetInteger(0,CHART_WINDOWS_TOTAL);
Print("Скрипт запущен на окне "+myWindow+
". Всего окон на графике "+ChartSymbol()+":",windowsTotal);
```

### Смотри также

[ChartPriceOnDropped](#), [ChartTimeOnDropped](#), [ChartXOnDropped](#), [ChartYOnDropped](#)

## ChartPriceOnDropped

Возвращает ценовую координату, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт.

```
double ChartPriceOnDropped();
```

### Возвращаемое значение

Значение типа [double](#).

### Пример:

```
double p=ChartPriceOnDropped();
Print("ChartPriceOnDropped() = ",p);
```

### Смотри также

[ChartXOnDropped](#), [ChartYOnDropped](#)

## ChartTimeOnDropped

Возвращает временную координату, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт.

```
datetime ChartTimeOnDropped();
```

### Возвращаемое значение

Значение типа [datetime](#).

### Пример:

```
datetime t=ChartTimeOnDropped();
Print("Script wasdropped on the "+t);
```

### Смотри также

[ChartXOnDropped](#), [ChartYOnDropped](#)

## ChartXOnDropped

Возвращает координату по оси X, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт.

```
int ChartXOnDropped();
```

### Возвращаемое значение

Значение координаты X.

### Примечание

Ось X направлена слева направо.

### Пример:

```
int X=ChartXOnDropped();
int Y=ChartYOnDropped();
Print("(X,Y) = ("+X+","+Y+")");
```

### Смотри также

[ChartWindowOnDropped](#), [ChartPriceOnDropped](#), [ChartTimeOnDropped](#)

## ChartYOnDropped

Возвращает координату по оси Y, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт.

```
int ChartYOnDropped();
```

### Возвращаемое значение

Значение координаты Y.

### Примечание

Ось Y направлена сверху вниз.

### Смотри также

[ChartWindowOnDropped](#), [ChartPriceOnDropped](#), [ChartTimeOnDropped](#)

## ChartSetSymbolPeriod

Меняет значения символа и периода указанного графика. Функция работает асинхронно, то есть отдает команду и не ждет окончания ее выполнения. Отданная команда поступает в очередь сообщений графика и выполняется только после обработки всех предыдущих команд.

```
bool ChartSetSymbolPeriod(
    long          chart_id,      // идентификатор графика
    string        symbol,        // имя символа
    ENUM_TIMEFRAMES period       // период
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*symbol*

[in] Символ графика. [NULL](#) означает символ текущего графика (к которому прикреплен эксперт)

*period*

[in] Период графика (таймфрейм). Может принимать одно из значений значений перечисления [ENUM\\_TIMEFRAMES](#). 0 означает период текущего графика.

### Возвращаемое значение

Возвращает true в случае удачного помещения команды в очередь графика, иначе false. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Смена символа/периода влечет за собой переинициализацию эксперта, прикрепленного к соответствующему графику.

Вызов `ChartSetSymbolPeriod` с тем же символом и таймфреймом можно использовать для обновления графика (аналогично команде `Refresh` в терминале). Обновление графика в свою очередь запускает перерасчет индикаторов, прикрепленных к нему. Таким образом, вы можете рассчитать индикатор на графике даже при отсутствии тиков (например, в выходные дни).

### Смотри также

[ChartSymbol](#), [ChartPeriod](#)

## ChartScreenShot

Функция обеспечивает скриншот указанного графика в его текущем состоянии в формате GIF, PNG или BMP в зависимости от указанного расширения.

```
bool ChartScreenShot(
    long          chart_id,           // идентификатор графика
    string        filename,          // имя файла
    int           width,             // ширина
    int           height,            // высота
    ENUM_ALIGN_MODE align_mode=ALIGN_RIGHT // тип выравнивания
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*filename*

[in] Имя файла скриншота. Не может превышать 63 символов. Скриншот помещается в директорию \Files.

*width*

[in] Ширина скриншота в пикселях

*height*

[in] Высота скриншота в пикселях

*align\_mode=ALIGN\_RIGHT*

[in] Режим вывода узкого скриншота. Значение перечисления [ENUM\\_ALIGN\\_MODE](#). ALIGN\_RIGHT означает выравнивание по правой границе (вывод с конца). ALIGN\_LEFT задает выравнивание по левой границе.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Если необходимо снять скриншот графика с определенной позиции, то необходимо сначала позиционировать график при помощи функции [ChartNavigate\(\)](#). Если горизонтальный размер скриншота меньше, чем окно графика, то выводится либо правая часть окна графика, либо левая часть, в зависимости от значения параметра *align\_mode*.

### Пример:

```
#property description "Советник демонстрирует создание серии скриншотов текущего графика"
#property description "с помощью функции ChartScreenShot(). Имя файла для удобства также задается макросами WIDTH и HEIGHT"
#property description "выводится на график. Высота и ширина рисунков задается макросами WIDTH и HEIGHT"

#define          WIDTH   800      // ширина рисунка для вызова ChartScreenShot()
#define          HEIGHT  600      // высота рисунка для вызова ChartScreenShot()
```

```

//--- input parameters
input int      pictures=5;      // количество рисунков в серии
int           mode=-1;          // -1 означает смещение к правому краю графика, 1 - к левому
int           bars_shift=300;    // количество баров при прокрутке графика функцией ChartNavigate()
//+-----+
//| Expert initialization function
//+-----+
void OnInit()
{
//--- отключим автопрокрутку графика
    ChartSetInteger(0,CHART_AUTOSCROLL,false);
//--- установим отступ правого края графика
    ChartSetInteger(0,CHART_SHIFT,true);
//--- установим отображение графика в виде свечей
    ChartSetInteger(0,CHART_MODE,CHART_CANDLES);
//---
//| Print("Подготовка советника к работе завершена");
}
//+-----+
//| Expert tick function
//+-----+
void OnTick()
{
//---

}
//+-----+
//| ChartEvent function
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
//--- вывод имени функции, времени вызова и идентификатора события
    Print(__FUNCTION__,TimeCurrent()," id=",id," mode=",mode);
//--- обработка события CHARTEVENT_CLICK ("Нажатие кнопки мышки на графике")
    if(id==CHARTEVENT_CLICK)
    {
//--- начальное смещение от края графика
        int pos=0;
//--- режим работы с левым краем графика
        if(mode>0)
        {
//--- прокрутим график к левому краю
            ChartNavigate(0,CHART_BEGIN,pos);
            for(int i=0;i<pictures;i++)
            {
//--- подготовим подпись на графике и имя для файла

```

```

string name="ChartScreenShot"+"CHART_BEGIN"+string(pos)+".gif";
//--- вывод имени на график в виде комментария
Comment(name);
//--- сохраним скриншот графика в папку каталог_терминала\MQL5\Files\
if(ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_LEFT))
    Print("Сохранили скриншот ",name);
//---
pos+=bars_shift;
//--- дадим пользователю время чтобы посмотреть на новый участок графика
Sleep(3000);
//--- прокрутим график от текущей позиции на bars_shift вправо
ChartNavigate(0,CHART_CURRENT_POS,bars_shift);
}
//--- смена режима на противоположный
mode*=-1;
}
else // режим работы с правым краем графика
{
//--- прокрутим график к правому краю
ChartNavigate(0,CHART_END,pos);
for(int i=0;i

```

#### Смотри также

[ChartNavigate\(\)](#), [Ресурсы](#)

## Торговые функции

Группа функций, предназначенных для управления торговой деятельностью.

Перед тем как приступить к изучению торговых функций платформы, необходимо создать четкое представление об основных терминах: ордер, сделка и позиция:

- Ордер - это распоряжение брокерской компании купить или продать финансовый инструмент. Различают два основных типа ордеров: рыночный и отложенный. Помимо них существуют специальные ордера Тейк Профит и Стоп Лосс.
- Сделка - факт покупки или продажи того или иного финансового инструмента. Покупка (Buy) происходит по цене спроса (Ask), а продажа (Sell) - по цене предложения (Bid). Сделка может быть совершена в результате исполнения рыночного ордера или срабатывания отложенного. Следует учитывать, что в некоторых случаях результатом исполнения ордера могут быть сразу несколько сделок.
- Позиция - это рыночное обязательство, количество купленных или проданных контрактов по финансовому инструменту. Длинная позиция (Long) - купленный в расчете на повышение цены финансовый инструмент, короткая (Short) - обязательство на его поставку в расчете на снижение цены в будущем.

Общая информация о торговых операциях доступна в [руководстве пользователя клиентского терминала](#).

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".

Разрешение или запрет на торговлю может зависеть от множества факторов, которые описаны в разделе "[Разрешение на торговлю](#)".

Функция	Действие
<a href="#">OrderCalcMargin</a>	Вычисляет размер маржи, необходимой для указанного типа ордера, в валюте счета
<a href="#">OrderCalcProfit</a>	Вычисляет размер прибыли на основании переданных параметров в валюте счета
<a href="#">OrderCheck</a>	Проверяет достаточность средств для совершения требуемой <a href="#">торговой операции</a> .
<a href="#">OrderSend</a>	Отправляет <a href="#">торговые запросы</a> на сервер
<a href="#">OrderSendAsync</a>	Отправляет асинхронно <a href="#">торговые запросы</a> без ожидания ответа торгового сервера
<a href="#">PositionsTotal</a>	Возвращает количество открытых позиций
<a href="#">PositionGetSymbol</a>	Возвращает символ соответствующей открытой позиции
<a href="#">PositionSelect</a>	Выбирает открытую позицию для дальнейшей работы с ней

<a href="#">PositionSelectByTicket</a>	Выбирает открытую позицию для дальнейшей работы с ней по указанному тикету
<a href="#">PositionGetDouble</a>	Возвращает запрошенное свойство открытой позиции (double)
<a href="#">PositionGetInteger</a>	Возвращает запрошенное свойство открытой позиции (datetime или int)
<a href="#">PositionGetString</a>	Возвращает запрошенное свойство открытой позиции (string)
<a href="#">PositionGetTicket</a>	Возвращает тикет позиции по индексу в списке открытых позиций
<a href="#">OrdersTotal</a>	Возвращает количество ордеров
<a href="#">OrderGetTicket</a>	Возвращает тикет соответствующего ордера
<a href="#">OrderSelect</a>	Выбирает ордер для дальнейшей работы с ним
<a href="#">OrderGetDouble</a>	Возвращает запрошенное свойство ордера (double)
<a href="#">OrderGetInteger</a>	Возвращает запрошенное свойство ордера (datetime или int)
<a href="#">OrderGetString</a>	Возвращает запрошенное свойство ордера (string)
<a href="#">HistorySelect</a>	Запрашивает историю сделок и ордеров за указанный период серверного времени
<a href="#">HistorySelectByPosition</a>	Запрашивает историю сделок и ордеров с указанным <b>идентификатором позиции</b> .
<a href="#">HistoryOrderSelect</a>	Выбирает в истории ордер для дальнейшей работы с ним
<a href="#">HistoryOrdersTotal</a>	Возвращает количество ордеров в истории
<a href="#">HistoryOrderGetTicket</a>	Возвращает тикет соответствующего ордера в истории
<a href="#">HistoryOrderGetDouble</a>	Возвращает запрошенное свойство ордера в истории (double)
<a href="#">HistoryOrderGetInteger</a>	Возвращает запрошенное свойство ордера в истории (datetime или int)
<a href="#">HistoryOrderGetString</a>	Возвращает запрошенное свойство ордера в истории (string)
<a href="#">HistoryDealSelect</a>	Выбирает в истории сделку для дальнейших обращений к ней через соответствующие функции

<a href="#"><u>HistoryDealsTotal</u></a>	Возвращает количество сделок в истории
<a href="#"><u>HistoryDealGetTicket</u></a>	Выбирает сделку для дальнейшей обработки и возвращает тикет сделки в истории
<a href="#"><u>HistoryDealGetDouble</u></a>	Возвращает запрошенное свойство сделки в истории (double)
<a href="#"><u>HistoryDealGetInteger</u></a>	Возвращает запрошенное свойство сделки в истории (datetime или int)
<a href="#"><u>HistoryDealGetString</u></a>	Возвращает запрошенное свойство сделки в истории (string)

## OrderCalcMargin

Вычисляет размер маржи, необходимой для указанного типа ордера на текущем счете и при текущем рыночном окружении без учета текущих отложенных ордеров и открытых позиций. Позволяет оценить размер маржи для планируемой торговой операции. Значение возвращается в валюте счета.

```
bool OrderCalcMargin(
    ENUM_ORDER_TYPE      action,           // тип ордера
    string               symbol,          // имя символа
    double               volume,          // объем
    double               price,           // цена открытия
    double&              margin           // переменная для получения значения маржи
);
```

### Параметры

*action*

[in] Тип ордера, может принимать значения из перечисления [ENUM\\_ORDER\\_TYPE](#).

*symbol*

[in] Имя финансового инструмента.

*volume*

[in] Объем торговой операции.

*price*

[in] Цена открытия.

*margin*

[out] Переменная, в которую будет записан необходимый размер маржи в случае успешного выполнения функции. Вычисление производится как если бы на текущем счете не было отложенных ордеров и открытых позиций. Значение маржи зависит от многих факторов и может меняться при изменении рыночного окружения.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Смотри также

[OrderSend\(\)](#), [Свойства ордеров](#), [Типы торговых операций](#)

## OrderCalcProfit

Вычисляет размер прибыли для текущего счета и рыночного окружения на основании переданных параметров. Предназначена для предварительной оценки результата торговой операции. Значение возвращается в валюте счета.

```
bool OrderCalcProfit(
    ENUM_ORDER_TYPE      action,           // тип ордера (ORDER_TYPE_BUY или ORDER_TYPE_SELL)
    string               symbol,          // имя символа
    double               volume,          // объем
    double               price_open,       // цена открытия
    double               price_close,      // цена закрытия
    double&              profit          // переменная для получения значения прибыли
);
```

### Параметры

*action*

[in] Тип ордера, может принимать одно из двух значений перечисления [ENUM\\_ORDER\\_TYPE](#): ORDER\_TYPE\_BUY или ORDER\_TYPE\_SELL.

*symbol*

[in] Имя финансового инструмента.

*volume*

[in] Объем торговой операции.

*price\_open*

[in] Цена открытия.

*price\_close*

[in] Цена закрытия.

*profit*

[out] Переменная, в которую будет записано вычисленное значение прибыли в случае успешного выполнения функции. Значение оценки прибыли зависит от многих факторов и может меняться при изменении рыночного окружения.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Если указан недопустимый тип ордера, функция вернет false. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Смотри также

[OrderSend\(\)](#), [Свойства ордеров](#), [Типы торговых операций](#)

## OrderCheck

Функция OrderCheck() проверяет достаточность средств для совершения требуемой [торговой операции](#). Результаты проверки помещаются в поля структуры [MqlTradeCheckResult](#).

```
bool OrderCheck(  
    MqlTradeRequest& request,          // структура запроса  
    MqlTradeCheckResult& result        // структура ответа  
) ;
```

### Параметры

*request*

[in] Указатель на структуру типа [MqlTradeRequest](#), которая описывает требуемое торговое действие.

*result*

[in,out] Указатель на структуру типа [MqlTradeCheckResult](#), в которую будет помещен результат проверки.

### Возвращаемое значение

В случае нехватки средств или ошибочно заполненных параметров функция возвращает false. В случае успешной базовой проверки структур (проверка указателей) возвращается true - это не является свидетельством того, что запрашиваемая торговая операция непременно выполнится успешно. Для получения подробного описания результата выполнения функции следует анализировать поля структуры *result*.

Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Смотри также

[OrderSend\(\)](#), [Типы торговых операций](#), [Структура торгового запроса](#), [Структура результатов проверки торгового запроса](#), [Структура результата торгового запроса](#)

## OrderSend

Функция OrderSend() предназначена для совершения [торговых операций](#) через отправку [запросов](#) на торговый сервер.

```
bool OrderSend(
    MqlTradeRequest& request,           // структура запроса
    MqlTradeResult& result             // структура ответа
);
```

### Параметры

*request*

[in] Указатель на структуру типа [MqlTradeRequest](#), описывающую торговое действие клиента.

*result*

[in,out] Указатель на структуру типа [MqlTradeResult](#), описывающую результат торговой операции в случае успешного выполнения (возврата true).

### Возвращаемое значение

В случае успешной базовой проверки структур (проверка указателей) возвращается true - это не свидетельствует об успешном выполнении торговой операции. Для получения более подробного описания результата выполнения функции следует анализировать поля структуры *result*.

### Примечание

Торговый запрос проходит несколько стадий проверок на торговом сервере. В первую очередь проверяется корректность заполнения всех необходимых полей параметра *request*, и при отсутствии ошибок сервер принимает ордер для дальнейшей обработки. При успешном принятии ордера торговым сервером функция OrderSend() возвращает значение true.

Рекомендуется самостоятельно проверить запрос перед отправкой его торговому серверу. Для проверки запроса существует функция [OrderCheck\(\)](#), которая не только проверит достаточность средств для совершения торговой операции, но и вернет в [результатах проверки торгового запроса](#) многие другие полезные параметры:

- [код возврата](#), который сообщит об ошибке в проверяемом запросе;
- значение баланса, которое будет после выполнения торговой операции;
- значение собственных средств, которое будет после выполнения торговой операции;
- значение плавающей прибыли, которое будет после выполнения торговой операции;
- размер маржи, необходимый для требуемой торговой операции;
- размер свободных собственных средств, которые останутся после выполнения требуемой торговой операции;
- уровень маржи, который установится после выполнения требуемой торговой операции;
- комментарий к коду ответа, описание ошибки.

При отправке рыночного ордера (`MqlTradeRequest.action=TRADE_ACTION DEAL`) успешный результат функции OrderSend() не означает, что ордер был выполнен (исполнены соответствующие сделки): true в этом случае означает только то, что ордер был успешно размещен в торговой системе для дальнейшего выполнения. Торговый сервер может в возвращаемой [структуре результата](#) *result* заполнить значения полей *deal* или *order*, если эти

данные будут ему известны в момент формирования ответа на вызов OrderSend(). В общем случае событие или события исполнения сделок, соответствующих ордеру, могут произойти уже после того, как будет отправлен ответ на вызов OrderSend(). Поэтому для любого типа торгового запроса при получении результата выполнения OrderSend() необходимо в первую очередь проверять код возврата торгового сервера *retcode* и код ответа внешней торговой системы *retcode\_external* (при необходимости), которые доступны в возвращаемой [структуре результата result](#).

Каждый принятый ордер хранится на торговом сервере в ожидании обработки до тех пор, пока не наступит одно из условий для его исполнения:

- истечение срока действия,
- появление встречного запроса,
- срабатывание ордера при поступлении цены исполнения,
- поступление запроса на отмену ордера.

В момент обработки ордера торговый сервер посыпает терминалу сообщение о наступлении торгового события [Trade](#), которое можно обработать функцией [OnTrade\(\)](#).

Результат исполнения торгового запроса на сервере, отправленного функцией OrderSend() можно отслеживать при помощи обработчика [OnTradeTransaction](#). При этом следует учитывать, что в результате исполнения одного торгового запроса обработчик OnTradeTransaction будет вызван несколько раз.

Например, при отсылке рыночного ордера на покупку, он обрабатывается, для счета создается соответствующий ордер на покупку, происходит исполнение ордера, его удаление из списка открытых, добавление в историю ордеров, далее добавляется соответствующая сделка в историю и создается новую позицию. Для каждого из этих событий будет вызвана функция OnTradeTransaction.

#### Пример:

```
//--- значения для ORDER_MAGIC
input long order_magic=55555;
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- убедимся, что счет является учебным
if(AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL)
{
    Alert("Работа скрипта на реальном счете запрещена!");
    return;
}
//--- установим либо удалим ордер
if(GetOrdersTotalByMagic(order_magic)==0)
{
//--- текущих ордеров нет - установим ордер
uint res=SendRandomPendingOrder(order_magic);
Print("Код возврата торгового сервера ",res);
}
```

```

else // ордера есть - удалим ордера
{
    DeleteAllOrdersByMagic(order_magic);
}

//---

}

//+-----+
//| Получает текущее количество ордеров с указанным ORDER_MAGIC |
//+-----+
int GetOrdersTotalByMagic(long const magic_number)
{
    ulong order_ticket;
    int total=0;
//--- пройдем по всем отложенным ордерам
    for(int i=0;i<OrdersTotal();i++)
        if((order_ticket=OrderGetTicket(i))>0)
            if(magic_number==OrderGetInteger(ORDER_MAGIC)) total++;
//---
    return(total);
}

//+-----+
//| Удаляет все отложенные ордера с указанным ORDER_MAGIC |
//+-----+
void DeleteAllOrdersByMagic(long const magic_number)
{
    ulong order_ticket;
//--- пройдем по всем отложенным ордерам
    for(int i=OrdersTotal()-1;i>=0;i--)
        if((order_ticket=OrderGetTicket(i))>0)
            //--- ордер с подходящим ORDER_MAGIC
            if(magic_number==OrderGetInteger(ORDER_MAGIC))
            {
                MqlTradeResult result={0};
                MqlTradeRequest request={0};
                request.order=order_ticket;
                request.action=TRADE_ACTION_REMOVE;
                OrderSend(request,result);
                //--- выведем в лог ответ сервера
                Print(__FUNCTION__," : ",result.comment," код ответа ",result.retcode);
            }
//---
    }
}

//+-----+
//| Установить случайным образом отложенный ордер |
//+-----+
uint SendRandomPendingOrder(long const magic_number)
{
//--- готовим запрос
    MqlTradeRequest request={0};
}

```

```

request.action=TRADE_ACTION_PENDING;           // установка отложенного ордера
request.magic=magic_number;                   // ORDER_MAGIC
request.symbol=_Symbol;                      // инструмент
request.volume=0.1;                          // объем в 0.1 лот
request.sl=0;                                // Stop Loss не указан
request.tp=0;                                // Take Profit не указан

//--- сформируем тип ордера
request.type=GetRandomType();                // тип ордера

//---сформируем цену для отложенного ордера
request.price=GetRandomPrice(request.type);   // цена для открытия

//--- отправим торговый приказ
MqlTradeResult result={0};
OrderSend(request,result);

//--- выведем в лог ответ сервера
Print(__FUNCTION__,":",result.comment);
if(result.retcode==10016) Print(result.bid,result.ask,result.price);

//--- вернем код ответа торгового сервера
return result.retcode;
}

//+-----+
//| Получить случайным образом тип отложенного типа |
//+-----+

ENUM_ORDER_TYPE GetRandomType()
{
    int t=MathRand()%4;
//--- 0<=t<4
    switch(t)
    {
        case(0):return(ORDER_TYPE_BUY_LIMIT);
        case(1):return(ORDER_TYPE_SELL_LIMIT);
        case(2):return(ORDER_TYPE_BUY_STOP);
        case(3):return(ORDER_TYPE_SELL_STOP);
    }
//--- недопустимое значение
    return(WRONG_VALUE);
}

//+-----+
//| Получить цену случайным образом |
//+-----+

double GetRandomPrice(ENUM_ORDER_TYPE type)
{
    int t=(int)type;
//--- уровень столов по символу
    int distance=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_STOPS_LEVEL);
//--- получим данные последнего тика
    MqlTick last_tick={0};
    SymbolInfoTick(_Symbol,last_tick);
//--- вычислим цену в соответствие с типом
    double price;
}

```

```
if(t==2 || t==5) // ORDER_TYPE_BUY_LIMIT или ORDER_TYPE_SELL_STOP
{
    price=last_tick.bid; // оттолкнемся от цены Bid
    price=price-(distance+(MathRand()%10)*5)*_Point;
}
else // ORDER_TYPE_SELL_LIMIT или ORDER_TYPE_BUY_STOP
{
    price=last_tick.ask; // оттолкнемся от цены Ask
    price=price+(distance+(MathRand()%10)*5)*_Point;
}
//---
return(price);
}
```

#### Смотри также

[Типы торговых операций](#), [Структура торгового запроса](#), [Структура результатов проверки торгового запроса](#), [Структура результата торгового запроса](#)

## OrderSendAsync

Функция OrderSendAsync() предназначена для проведения асинхронных [торговых операций](#) без ожидания ответа торгового сервера на отправленный [запрос](#). Функция предназначена для высокочастотной торговли, когда по условиям торгового алгоритма недопустимо терять время на ожидание ответа от сервера.

```
bool OrderSendAsync(
    MqlTradeRequest& request,           // структура запроса
    MqlTradeResult& result            // структура ответа
);
```

### Параметры

*request*

[in] Указатель на структуру типа [MqlTradeRequest](#), описывающую торговое действие клиента.

*result*

[in,out] Указатель на структуру типа [MqlTradeResult](#), описывающую результат торговой операции в случае успешного выполнения функции (при возврате true).

### Возвращаемое значение

Возвращает true по факту отправки запроса на торговый сервер. Если запрос не был отправлен, возвращается false. При успешном выполнении функции в переменной *result* код ответа содержит значение [TRADE\\_RETCODE\\_PLACED](#) (код 10008) - "ордер размещен". Успешное выполнение означает только факт отсылки, но не даёт никакой гарантии, что запрос дошел до торгового сервера и был принят для обработки. Торговый сервер при обработке полученного запроса отправляет клиентскому терминалу ответное сообщение об изменении текущего состояния позиций, ордеров и сделок, которое приводит к генерации события [Trade](#).

Результат исполнения торгового запроса на сервере, отправленного функцией OrderSendAsync() можно отслеживать при помощи обработчика [OnTradeTransaction](#). При этом следует учитывать, что в результате исполнения одного торгового запроса обработчик OnTradeTransaction будет вызван несколько раз.

Например, при отсылке рыночного ордера на покупку, он обрабатывается, для счета создается соответствующий ордер на покупку, происходит исполнение ордера, его удаление из списка открытых, добавление в историю ордеров, далее добавляется соответствующая сделка в историю и создается новую позицию. Для каждого из этих событий будет вызвана функция OnTradeTransaction. Для получения подробной информации следует анализировать параметры этой функции:

- *trans* - в данный параметр передается структура [MqlTradeTransaction](#), описывающая торговую транзакцию, примененную к торговому счету;
- *request* - в данный параметр передается структура [MqlTradeRequest](#), описывающая торговый запрос, в результате которого совершена торговая транзакция;
- *result* - в данный параметр передается структура [MqlTradeResult](#), описывающая результат исполнения торгового запроса.

### Примечание

Функция по назначению и параметрам аналогична [OrderSend\(\)](#), но в отличие от неё является асинхронной версией, то есть не приостанавливает работу программы в ожидании результата её

выполнения. Сравнить скорость торговых операций этих двух функций можно с помощью приведенного в примере советника.

### Пример:

```
#property description "Эксперт для отправки торговых запросов "
                     " с использованием функции OrderSendAsync()\r\n"
#property description "Показана обработка торговых событий с помощью"
                     " функций-обработчиков OnTrade() и OnTradeTransaction()\r\n"
#property description "В параметрах эксперта можно задать Magic Number"
                     " (уникальный идентификатор) "
#property description "и режим вывода сообщений в журнал \"Эксперты\". По умолчанию вывод
//--- input parameters
input int MagicNumber=1234567;           // Идентификатор эксперта
input bool DescriptionModeFull=true;        // Режим детального вывода
//--- переменная для использования в вызове HistorySelect()
datetime history_start;
//+-----+
//| Expert initialization function          |
//+-----+
int OnInit()
{
//--- проверим разрешение на автотрейдинг
if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
{
    Alert("Автотрейдинг в терминале запрещен, эксперт будет выгружен.");
    ExpertRemove();
    return(-1);
}
//--- на реальном счете торговать нельзя
if(AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL)
{
    Alert("Советнику запрещено торговать на реальном счете!");
    ExpertRemove();
    return(-2);
}
//--- можно ли торговать на данном счете (под инвест-паролем нельзя, например)
if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED))
{
    Alert("Торговля на данном счете запрещена");
    ExpertRemove();
    return(-3);
}
//--- запомним время запуска эксперта для получения торговой истории
history_start=TimeCurrent();
//---
CreateBuySellButtons();
return(INIT_SUCCEEDED);
}
```

```

//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
    //--- удалим за собой графические объекты
    ObjectDelete(0,"Buy");
    ObjectDelete(0,"Sell");
//---
}

//+-----+
//| TradeTransaction function
//+-----+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
{
    //--- заголовок по имени функции-обработчика торгового события
    Print("=> ",__FUNCTION__," at ",TimeToString(TimeCurrent(),TIME_SECONDS));
    //--- получим тип транзакции в виде значения перечисления
    ENUM_TRADE_TRANSACTION_TYPE type=trans.type;
    //--- если транзакция является результатом обработки запроса
    if(type==TRADE_TRANSACTION_REQUEST)
    {
        //---выведем название транзакции
        Print(EnumToString(type));
        //--- затем выведем строковое описание обработанного запроса
        Print("-----RequestDescription\r\n",
              RequestDescription(request,DescriptionModeFull));
        //--- и выведем описание результата запроса
        Print("----- ResultDescription\r\n",
              TradeResultDescription(result,DescriptionModeFull));
    }
    else // для транзакций другого типа выведем полное описание транзакции
    {
        Print("----- TransactionDescription\r\n",
              TransactionDescription(trans,DescriptionModeFull));
    }
//---
}

//+-----+
//| Trade function
//+-----+
void OnTrade()
{
    //--- статические члены для хранения состояния торгового счета
    static int prev_positions=0,prev_orders=0,prev_deals=0,prev_history_orders=0;
    //--- запросим торговую историю
    bool update=HistorySelect(history_start,TimeCurrent());
}

```

```

PrintFormat("HistorySelect(%s , %s) = %s",
            TimeToString(history_start), TimeToString(TimeCurrent()), (string)update)
//--- заголовок по имени функции-обработчика торгового события
Print("=> ", __FUNCTION__, " at ", TimeToString(TimeCurrent(), TIME_SECONDS));
//--- выведем имя обработчика и количество ордеров на момент обработки
int curr_positions=PositionsTotal();
int curr_orders=OrdersTotal();
int curr_deals=HistoryOrdersTotal();
int curr_history_orders=HistoryDealsTotal();
//--- выводим количество ордеров, позиций, сделок, а также изменение в скобках
PrintFormat("PositionsTotal() = %d (%+d)",
            curr_positions, (curr_positions-prev_positions));
PrintFormat("OrdersTotal() = %d (%+d)",
            curr_orders, curr_orders-prev_orders);
PrintFormat("HistoryOrdersTotal() = %d (%+d)",
            curr_deals, curr_deals-prev_deals);
PrintFormat("HistoryDealsTotal() = %d (%+d)",
            curr_history_orders, curr_history_orders-prev_history_orders);
//--- вставка разрыва строк для удобного чтения Журнала
Print("");
//--- запомним состояние счета
prev_positions=curr_positions;
prev_orders=curr_orders;
prev_deals=curr_deals;
prev_history_orders=curr_history_orders;
//---
}
//-----+
//| ChartEvent function
//-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
//--- обработка события CHARTEVENT_CLICK ("Нажатие кнопки мышки на графике")
if(id==CHARTEVENT_OBJECT_CLICK)
{
    Print("=> ", __FUNCTION__, ": sparam = ", sparam);
    //--- минимальный объем для сделки
    double volume_min=SymbolInfoDouble(_Symbol, SYMBOL_VOLUME_MIN);
    //--- если нажата кнопка "Buy", то покупаем
    if(sparam=="Buy")
    {
        PrintFormat("Buy %s %G lot", _Symbol, volume_min);
        BuyAsync(volume_min);
        //--- отожмем нажатую кнопку обратно
        ObjectSetInteger(0, "Buy", OBJPROP_STATE, false);
    }
}

```

```

//--- если нажата кнопка "Sell", то продаем
if(sparam=="Sell")
{
    PrintFormat("Sell %s %G lot",_Symbol,volume_min);
    SellAsync(volume_min);
    //--- отожмем нажатую кнопку обратно
    ObjectSetInteger(0,"Sell",OBJPROP_STATE,false);
}
ChartRedraw();
}

//---

}

//+-----+
//| Возвращает текстовое описание транзакции |
//+-----+
string TransactionDescription(const MqlTradeTransaction &trans,
                               const bool detailed=true)
{
//--- подготовим строку для возврата из функции
string desc=EnumToString(trans.type)+"\r\n";
//--- в детальном режиме добавим максимум информации
if(detailed)
{
    desc+="Symbol: "+trans.symbol+"\r\n";
    desc+="Deal ticket: "+(string)trans.deal+"\r\n";
    desc+="Deal type: "+EnumToString(trans.deal_type)+"\r\n";
    desc+="Order ticket: "+(string)trans.order+"\r\n";
    desc+="Order type: "+EnumToString(trans.order_type)+"\r\n";
    desc+="Order state: "+EnumToString(trans.order_state)+"\r\n";
    desc+="Order time type: "+EnumToString(trans.time_type)+"\r\n";
    desc+="Order expiration: "+TimeString(trans.time_expiration)+"\r\n";
    desc+="Price: "+StringFormat("%G",trans.price)+"\r\n";
    desc+="Price trigger: "+StringFormat("%G",trans.price_trigger)+"\r\n";
    desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"\r\n";
    desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"\r\n";
    desc+="Volume: "+StringFormat("%G",trans.volume)+"\r\n";
}
//--- вернем полученную строку
return desc;
}

//+-----+
//| Возвращает текстовое описание торгового запроса |
//+-----+
string RequestDescription(const MqlTradeRequest &request,
                           const bool detailed=true)
{
//--- подготовим строку для возврата из функции
string desc=EnumToString(request.action)+"\r\n";
//--- в детальном режиме добавим максимум информации
}

```

```

if(detailed)
{
    desc+="Symbol: "+request.symbol+"\r\n";
    desc+="Magic Number: "+StringFormat("%d",request.magic)+"\r\n";
    desc+="Order ticket: "+(string)request.order+"\r\n";
    desc+="Order type: "+EnumToString(request.type)+"\r\n";
    desc+="Order filling: "+EnumToString(request.type_filling)+"\r\n";
    desc+="Order time type: "+EnumToString(request.type_time)+"\r\n";
    desc+="Order expiration: "+TimeToString(request.expiration)+"\r\n";
    desc+="Price: "+StringFormat("%G",request.price)+"\r\n";
    desc+="Deviation points: "+StringFormat("%G",request.deviation)+"\r\n";
    desc+="Stop Loss: "+StringFormat("%G",request.sl)+"\r\n";
    desc+="Take Profit: "+StringFormat("%G",request.tp)+"\r\n";
    desc+="Stop Limit: "+StringFormat("%G",request.stoplimit)+"\r\n";
    desc+="Volume: "+StringFormat("%G",request.volume)+"\r\n";
    desc+="Comment: "+request.comment+"\r\n";
}
//--- вернем полученную строку
return desc;
}

//-----+
//| Возвращает текстовое описание результата обработки запроса |
//-----+
string TradeResultDescription(const MqlTradeResult &result,
                               const bool detailed=true)
{
//--- подготовим строку для возврата из функции
string desc="Retcode "+(string)result.retcode+"\r\n";
//--- в детальном режиме добавим максимум информации
if(detailed)
{
    desc+="Request ID: "+StringFormat("%d",result.request_id)+"\r\n";
    desc+="Order ticket: "+(string)result.order+"\r\n";
    desc+="Deal ticket: "+(string)result.deal+"\r\n";
    desc+="Volume: "+StringFormat("%G",result.volume)+"\r\n";
    desc+="Price: "+StringFormat("%G",result.price)+"\r\n";
    desc+="Ask: "+StringFormat("%G",result.ask)+"\r\n";
    desc+="Bid: "+StringFormat("%G",result.bid)+"\r\n";
    desc+="Comment: "+result.comment+"\r\n";
}
//--- вернем полученную строку
return desc;
}

//-----+
//| Создает две кнопки для покупки и продажи |
//-----+
void CreateBuySellButtons()
{
//--- проверим наличие объекта с именем "Buy"

```

```

if(ObjectFind(0, "Buy")>=0)
{
    //--- если найденный объект не является кнопкой, удалим его
    if(ObjectGetInteger(0, "Buy", OBJPROP_TYPE) !=OBJ_BUTTON)
        ObjectDelete(0, "Buy");
}
else
    ObjectCreate(0, "Buy", OBJ_BUTTON,0,0,0); // создадим кнопку "Buy"
//--- настроим кнопку "Buy"
ObjectSetInteger(0, "Buy", OBJPROP_CORNER,CORNER_RIGHT_UPPER);
ObjectSetInteger(0, "Buy", OBJPROP_XDISTANCE,100);
ObjectSetInteger(0, "Buy", OBJPROP_YDISTANCE,50);
ObjectSetInteger(0, "Buy", OBJPROP_XSIZE,70);
ObjectSetInteger(0, "Buy", OBJPROP_YSIZE,30);
ObjectSetString(0, "Buy", OBJPROP_TEXT,"Buy");
ObjectSetInteger(0, "Buy", OBJPROP_COLOR,clrRed);
//--- проверим наличие объекта с именем "Sell"
if(ObjectFind(0, "Sell")>=0)
{
    //--- если найденный объект не является кнопкой, удалим его
    if(ObjectGetInteger(0, "Sell", OBJPROP_TYPE) !=OBJ_BUTTON)
        ObjectDelete(0, "Sell");
}
else
    ObjectCreate(0, "Sell", OBJ_BUTTON,0,0,0); // создадим кнопку "Sell"
//--- настроим кнопку "Sell"
ObjectSetInteger(0, "Sell", OBJPROP_CORNER,CORNER_RIGHT_UPPER);
ObjectSetInteger(0, "Sell", OBJPROP_XDISTANCE,100);
ObjectSetInteger(0, "Sell", OBJPROP_YDISTANCE,100);
ObjectSetInteger(0, "Sell", OBJPROP_XSIZE,70);
ObjectSetInteger(0, "Sell", OBJPROP_YSIZE,30);
ObjectSetString(0, "Sell", OBJPROP_TEXT,"Sell");
ObjectSetInteger(0, "Sell", OBJPROP_COLOR,clrBlue);
//--- принудительно обновим график, чтобы кнопки отрисовались немедленно
ChartRedraw();
//---
}

//-----+
//| Покупка через асинхронную функцию OrderSendAsync() |+
//-----+
void BuyAsync(double volume)
{
//--- подготовим запрос
MqlTradeRequest req={0};
req.action      =TRADE_ACTION_DEAL;
req.symbol      =_Symbol;
req.magic       =MagicNumber;
req.volume      =0.1;
req.type        =ORDER_TYPE_BUY;
}

```

```

req.price      =SymbolInfoDouble(req.symbol,SYMBOL_ASK);
req.deviation  =10;
req.comment    ="Buy using OrderSendAsync()";
MqlTradeResult res={0};
if(!OrderSendAsync(req,res))
{
    Print(__FUNCTION__,": ошибка ",GetLastError()," , retcode = ",res.retcode);
}
//---
}
//+-----+
// | Продажа через асинхронную функцию OrderSendAsync()
//+-----+
void SellAsync(double volume)
{
//--- подготовим запрос
MqlTradeRequest req={0};
req.action      =TRADE_ACTION_DEAL;
req.symbol      =_Symbol;
req.magic       =MagicNumber;
req.volume      =0.1;
req.type        =ORDER_TYPE_SELL;
req.price       =SymbolInfoDouble(req.symbol,SYMBOL_BID);
req.deviation   =10;
req.comment     ="Sell using OrderSendAsync()";
MqlTradeResult res={0};
if(!OrderSendAsync(req,res))
{
    Print(__FUNCTION__,": ошибка ",GetLastError()," , retcode = ",res.retcode);
}
//---
}
//+-----+

```

Пример вывода сообщений в журнал "Эксперты":

```

12:52:52 ExpertAdvisor (EURUSD,H1) => OnChartEvent: sparam = Sell
12:52:52 ExpertAdvisor (EURUSD,H1) Sell EURUSD 0.01 lot
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTradeTransaction at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) TRADE_TRANSACTION_REQUEST
12:52:52 ExpertAdvisor (EURUSD,H1) -----RequestDescription
12:52:52 ExpertAdvisor (EURUSD,H1) TRADE_ACTION_DEAL
12:52:52 ExpertAdvisor (EURUSD,H1) Symbol: EURUSD
12:52:52 ExpertAdvisor (EURUSD,H1) Magic Number: 1234567
12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998
12:52:52 ExpertAdvisor (EURUSD,H1) Order type: ORDER_TYPE_SELL
12:52:52 ExpertAdvisor (EURUSD,H1) Order filling: ORDER_FILLING_FOK
12:52:52 ExpertAdvisor (EURUSD,H1) Order time type: ORDER_TIME_GTC
12:52:52 ExpertAdvisor (EURUSD,H1) Order expiration: 1970.01.01 00:00

```

```

12:52:52 ExpertAdvisor (EURUSD,H1) Price: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Deviation points: 10
12:52:52 ExpertAdvisor (EURUSD,H1) Stop Loss: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Take Profit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Stop Limit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0.1
12:52:52 ExpertAdvisor (EURUSD,H1) Comment: Sell using OrderSendAsync()
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) ----- ResultDescription
12:52:52 ExpertAdvisor (EURUSD,H1) Retcode 10009
12:52:52 ExpertAdvisor (EURUSD,H1) Request ID: 2
12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998
12:52:52 ExpertAdvisor (EURUSD,H1) Deal ticket: 15048668
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0.1
12:52:52 ExpertAdvisor (EURUSD,H1) Price: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Ask: 1.29319
12:52:52 ExpertAdvisor (EURUSD,H1) Bid: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Comment:
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) HistorySelect( 09:34 , 09:52) = true
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTrade at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) PositionsTotal() = 1 (+1)
12:52:52 ExpertAdvisor (EURUSD,H1) OrdersTotal() = 0 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryOrdersTotal() = 2 (+2)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryDealsTotal() = 2 (+2)
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTradeTransaction at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) ----- TransactionDescription
12:52:52 ExpertAdvisor (EURUSD,H1) TRADE_TRANSACTION_ORDER_ADD
12:52:52 ExpertAdvisor (EURUSD,H1) Symbol: EURUSD
12:52:52 ExpertAdvisor (EURUSD,H1) Deal ticket: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Deal type: DEAL_TYPE_BUY
12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998
12:52:52 ExpertAdvisor (EURUSD,H1) Order type: ORDER_TYPE_SELL
12:52:52 ExpertAdvisor (EURUSD,H1) Order state: ORDER_STATE_STARTED
12:52:52 ExpertAdvisor (EURUSD,H1) Order time type: ORDER_TIME_GTC
12:52:52 ExpertAdvisor (EURUSD,H1) Order expiration: 1970.01.01 00:00
12:52:52 ExpertAdvisor (EURUSD,H1) Price: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Price trigger: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Stop Loss: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Take Profit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0.1
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTradeTransaction at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) ----- TransactionDescription
12:52:52 ExpertAdvisor (EURUSD,H1) TRADE_TRANSACTION_ORDER_DELETE
12:52:52 ExpertAdvisor (EURUSD,H1) Symbol: EURUSD
12:52:52 ExpertAdvisor (EURUSD,H1) Deal ticket: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Deal type: DEAL_TYPE_BUY

```

```

12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998
12:52:52 ExpertAdvisor (EURUSD,H1) Order type: ORDER_TYPE_SELL
12:52:52 ExpertAdvisor (EURUSD,H1) Order state: ORDER_STATE_STARTED
12:52:52 ExpertAdvisor (EURUSD,H1) Order time type: ORDER_TIME_GTC
12:52:52 ExpertAdvisor (EURUSD,H1) Order expiration: 1970.01.01 00:00
12:52:52 ExpertAdvisor (EURUSD,H1) Price: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Price trigger: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Stop Loss: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Take Profit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0.1
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) HistorySelect( 09:34 , 09:52) = true
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTrade at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) PositionsTotal() = 1 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) OrdersTotal() = 0 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryOrdersTotal() = 2 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryDealsTotal() = 2 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTradeTransaction at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) ----- TransactionDescription
12:52:52 ExpertAdvisor (EURUSD,H1) TRADE_TRANSACTION_HISTORY_ADD
12:52:52 ExpertAdvisor (EURUSD,H1) Symbol: EURUSD
12:52:52 ExpertAdvisor (EURUSD,H1) Deal ticket: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Deal type: DEAL_TYPE_BUY
12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998
12:52:52 ExpertAdvisor (EURUSD,H1) Order type: ORDER_TYPE_SELL
12:52:52 ExpertAdvisor (EURUSD,H1) Order state: ORDER_STATE_FILLED
12:52:52 ExpertAdvisor (EURUSD,H1) Order time type: ORDER_TIME_GTC
12:52:52 ExpertAdvisor (EURUSD,H1) Order expiration: 1970.01.01 00:00
12:52:52 ExpertAdvisor (EURUSD,H1) Price: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Price trigger: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Stop Loss: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Take Profit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) HistorySelect( 09:34 , 09:52) = true
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTrade at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) PositionsTotal() = 1 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) OrdersTotal() = 0 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryOrdersTotal() = 2 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryDealsTotal() = 2 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTradeTransaction at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) ----- TransactionDescription
12:52:52 ExpertAdvisor (EURUSD,H1) TRADE_TRANSACTION_DEAL_ADD
12:52:52 ExpertAdvisor (EURUSD,H1) Symbol: EURUSD
12:52:52 ExpertAdvisor (EURUSD,H1) Deal ticket: 15048668
12:52:52 ExpertAdvisor (EURUSD,H1) Deal type: DEAL_TYPE_SELL
12:52:52 ExpertAdvisor (EURUSD,H1) Order ticket: 16361998

```

```
12:52:52 ExpertAdvisor (EURUSD,H1) Order type: ORDER_TYPE_BUY
12:52:52 ExpertAdvisor (EURUSD,H1) Order state: ORDER_STATE_STARTED
12:52:52 ExpertAdvisor (EURUSD,H1) Order time type: ORDER_TIME_GTC
12:52:52 ExpertAdvisor (EURUSD,H1) Order expiration: 1970.01.01 00:00
12:52:52 ExpertAdvisor (EURUSD,H1) Price: 1.29313
12:52:52 ExpertAdvisor (EURUSD,H1) Price trigger: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Stop Loss: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Take Profit: 0
12:52:52 ExpertAdvisor (EURUSD,H1) Volume: 0.1
12:52:52 ExpertAdvisor (EURUSD,H1)
12:52:52 ExpertAdvisor (EURUSD,H1) HistorySelect( 09:34 , 09:52) = true
12:52:52 ExpertAdvisor (EURUSD,H1) => OnTrade at 09:52:53
12:52:52 ExpertAdvisor (EURUSD,H1) PositionsTotal() = 1 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) OrdersTotal() = 0 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryOrdersTotal() = 2 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1) HistoryDealsTotal() = 2 (+0)
12:52:52 ExpertAdvisor (EURUSD,H1)
```

## PositionsTotal

Возвращает количество открытых позиций.

```
int PositionsTotal();
```

### Возвращаемое значение

Значение типа [int](#).

### Примечание

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций.

### Смотри также

[PositionGetSymbol\(\)](#), [PositionSelect\(\)](#), [Свойства позиций](#)

## PositionGetSymbol

Возвращает символ соответствующей открытой позиции и автоматически выбирает позицию для дальнейшей работы с ней при помощи функций [PositionGetDouble](#), [PositionGetInteger](#), [PositionGetString](#).

```
string PositionGetSymbol(  
    int index // номер в списке позиций  
) ;
```

### Параметры

*index*

[in] Номер позиции в списке открытых позиций.

### Возвращаемое значение

Значение типа [string](#). Если позиция не найдена, то вернется пустая строка. Для получения [кода ошибки](#) нужно вызвать функцию [GetLastError\(\)](#).

### Примечание

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций.

### Смотри также

[PositionsTotal\(\)](#), [PositionSelect\(\)](#), [Свойства позиций](#)

## PositionSelect

Выбирает открытую позицию для дальнейшей работы с ней. Возвращает true при успешном завершении функции. Возвращает false при неудачном завершении функции. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

```
bool PositionSelect(
    string symbol      // имя инструмента
);
```

### Параметры

*symbol*  
 [in] Наименование финансового инструмента.

### Возвращаемое значение

Значение типа bool.

### Примечание

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому символу в любой момент времени может быть открыта только одна позиция, которая является результатом одной или более  сделок. Не следует путать между собой позиции и действующие отложенные ордера, которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций. В этом случае, PositionSelect выберет позицию с наименьшим тикетом.

Функция PositionSelect() копирует данные о позиции в программное окружение, и последующие вызовы [PositionGetDouble\(\)](#), [PositionGetInteger\(\)](#) и [PositionGetString\(\)](#) возвращают ранее скопированные данные. Это означает, что самой позиции может уже и не быть (или же она изменилась по объему, направлению и т.д.), а данные этой позиции можно еще получать. Для гарантированного получения свежих данных о позиции рекомендуется вызывать функцию PositionSelect() непосредственно перед обращением за ними.

### Смотри также

[PositionGetSymbol\(\)](#), [PositionsTotal\(\)](#), [Свойства позиций](#)

## PositionSelectByTicket

Выбирает открытую позицию для дальнейшей работы с ней по указанному тикету. Возвращает true при успешном завершении функции. Возвращает false при неудачном завершении функции. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

```
bool PositionSelectByTicket(  
    ulong ticket // тикет позиции  
) ;
```

### Параметры

*ticket*

[in] Тикет позиции.

### Возвращаемое значение

Значение типа bool.

### Примечание

Функция PositionSelectByTicket() копирует данные о позиции в программное окружение, и последующие вызовы [PositionGetDouble\(\)](#), [PositionGetInteger\(\)](#) и [PositionGetString\(\)](#) возвращают ранее скопированные данные. Это означает, что самой позиции может уже и не быть (или же она изменилась по объему, направлению и т.д.), а данные этой позиции можно еще получать. Для гарантированного получения свежих данных о позиции рекомендуется вызывать функцию PositionSelectByTicket() непосредственно перед обращением за ними.

### Смотри также

[PositionGetSymbol\(\)](#), [PositionsTotal\(\)](#), [Свойства позиций](#)

## PositionGetDouble

Функция возвращает запрошенное свойство открытой позиции, предварительно выбранной при помощи функции [PositionGetSymbol](#) или [PositionSelect](#). Свойство позиции должно быть типа double. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
double PositionGetDouble(
    ENUM_POSITION_PROPERTY_DOUBLE property_id           // идентификатор свойства
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool PositionGetDouble(
    ENUM_POSITION_PROPERTY_DOUBLE property_id,           // идентификатор свойства
    double&                      double_var            // сюда примем значение свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства позиции. Значение может быть одним из значений перечисления [ENUM\\_POSITION\\_PROPERTY\\_DOUBLE](#).

*double\_var*

[out] Переменная типа double, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [double](#). В случае неудачного выполнения возвращает 0.

### Примечание

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций.

Для гарантированного получения свежих данных о позиции рекомендуется вызывать функцию [PositionSelect\(\)](#) непосредственно перед обращением за ними.

### Смотри также

[PositionGetSymbol\(\)](#), [PositionSelect\(\)](#), [Свойства позиций](#)

## PositionGetInteger

Функция возвращает запрошенное свойство открытой позиции, предварительно выбранной при помощи функции [PositionGetSymbol](#) или [PositionSelect](#). Свойство позиции должно быть типа `datetime`, `int`. Существует 2 варианта функции.

- Непосредственно возвращает значение свойства.

```
long PositionGetInteger(
    ENUM_POSITION_PROPERTY_INTEGER property_id           // идентификатор свойства
);
```

- Возвращает `true` или `false` в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool PositionGetInteger(
    ENUM_POSITION_PROPERTY_INTEGER property_id,          // идентификатор свойства
    long&                      long_var                // сюда примем значение свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства позиции. Значение может быть одним из значений перечисления [ENUM\\_POSITION\\_PROPERTY\\_INTEGER](#).

*long\_var*

[out] Переменная типа `long`, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [long](#). В случае неудачного выполнения возвращает 0.

### Примечание

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций.

Для гарантированного получения свежих данных о позиции рекомендуется вызывать функцию [PositionSelect\(\)](#) непосредственно перед обращением за ними.

### Пример:

```
//+-----+
//| Trade function
//+-----+
void OnTrade()
```

```
{  
//--- проверим наличие позиции и выведем время её изменения  
if(PositionSelect(_Symbol))  
{  
//--- получим идентификатор позиции для дальнейшей работы с ней  
ulong position_ID=PositionGetInteger(POSITION_IDENTIFIER);  
Print(_Symbol, " position #",position_ID);  
//--- получим время образования позиции в миллисекундах с 01.01.1970  
long create_time_msc=PositionGetInteger(POSITION_TIME_MSC);  
PrintFormat("Position #%d POSITION_TIME_MSC = %i64 milliseconds => %s",position_ID,  
create_time_msc,TimeToString(create_time_msc/1000));  
//--- получим время последнего изменения позиции в секундах с 01.01.1970  
long update_time_sec=PositionGetInteger(POSITION_TIME_UPDATE);  
PrintFormat("Position #%d POSITION_TIME_UPDATE = %i64 seconds => %s",  
position_ID,update_time_sec,TimeToString(update_time_sec));  
//--- получим время последнего изменения позиции в миллисекундах с 01.01.1970  
long update_time_msc=PositionGetInteger(POSITION_TIME_UPDATE_MSC);  
PrintFormat("Position #%d POSITION_TIME_UPDATE_MSC = %i64 milliseconds => %s",  
position_ID,update_time_msc,TimeToString(update_time_msc/1000));  
}  
//---  
}
```

#### Смотри также

[PositionGetSymbol\(\)](#), [PositionSelect\(\)](#), [Свойства позиций](#)

## PositionGetString

Функция возвращает запрошенное свойство открытой позиции, предварительно выбранной при помощи функции [PositionGetSymbol](#) или [PositionSelect](#). Свойство позиции должно быть типа `string`. Существует 2 варианта функции.

- Непосредственно возвращает значение свойства.

```
string PositionGetString(
    ENUM_POSITION_PROPERTY_STRING property_id           // идентификатор свойства
);
```

- Возвращает `true` или `false` в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool PositionGetString(
    ENUM_POSITION_PROPERTY_STRING property_id,          // идентификатор свойства
    string&                      string_var           // сюда примем значение свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства позиции. Значение может быть одним из значений перечисления [ENUM\\_POSITION\\_PROPERTY\\_STRING](#).

*string\_var*

[out] Переменная типа `string`, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [string](#). В случае неудачного выполнения возвращает пустую строку.

### Примечание

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций.

Для гарантированного получения свежих данных о позиции рекомендуется вызывать функцию [PositionSelect\(\)](#) непосредственно перед обращением за ними.

### Смотри также

[PositionGetSymbol\(\)](#), [PositionSelect\(\)](#), [Свойства позиций](#)

## PositionGetTicket

Функция возвращает тикет позиции по индексу в списке открытых позиций и автоматически выбирает эту позицию для дальнейшей работы с ней при помощи функций [PositionGetDouble](#), [PositionGetInteger](#), [PositionGetString](#).

```
ulong PositionGetTicket(  
    int index // номер в списке позиций  
) ;
```

### Параметры

*index*

[in] Индекс позиции в списке открытых позиций, начиная с 0.

### Возвращаемое значение

Тикет позиции. В случае неудачного выполнения возвращает 0.

### Примечание

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций.

Для гарантированного получения свежих данных о позиции рекомендуется вызывать функцию [PositionSelect\(\)](#) непосредственно перед обращением за ними.

### Смотри также

[PositionGetSymbol\(\)](#), [PositionSelect\(\)](#), [Свойства позиций](#)

## OrdersTotal

Возвращает количество действующих ордеров.

```
int OrdersTotal();
```

### Возвращаемое значение

Значение типа [int](#).

### Примечание

Не следует путать между собой действующие [отложенные ордера](#) и позиции, которые также отображаются на вкладке "Торговля" в панели "Инструменты". Ордер - это распоряжение на проведение [торговой операции](#), а позиция является результатом одной или нескольких [сделок](#).

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций.

### Смотри также

[OrderSelect\(\)](#), [OrderGetTicket\(\)](#), [Свойства ордеров](#)

## OrderGetTicket

Возвращает тикет соответствующего ордера и автоматически выбирает ордер для дальнейшей работы с ним при помощи функций.

```
ulong OrderGetTicket(
    int index // номер в списке ордеров
);
```

### Параметры

*index*

[in] Номер ордера в списке текущих ордеров.

### Возвращаемое значение

Значение типа [ulong](#). В случае неудачного выполнения возвращает 0.

### Примечание

Не следует путать между собой действующие [отложенные ордера](#) и позиции, которые также отображаются на вкладке "Торговля" в панели "Инструменты". Ордер - это распоряжение на проведение [торговой операции](#), а позиция является результатом одной или нескольких [сделок](#).

При "неттингом" учите позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций.

Функция OrderGetTicket() копирует данные об ордере в программное окружение, и последующие вызовы [OrderGetDouble\(\)](#), [OrderGetInteger\(\)](#), [OrderGetString\(\)](#) возвращают ранее скопированные данные. Это означает, что самого ордера может уже и не быть (или же в нем изменились цена открытия, уровни Stop Loss / Take Profit или момент истечения), а данные этому ордеру можно еще получать. Для гарантированного получения свежих данных об ордере рекомендуется вызывать функцию OrderGetTicket() непосредственно перед обращением за ними.

### Пример:

```
void OnStart()
{
//--- переменные для получения значений из свойств ордера
    ulong ticket;
    double open_price;
    double initial_volume;
    datetime time_setup;
    string symbol;
    string type;
    long order_magic;
    long positionID;
//--- количество текущих отложенных ордеров
```

```
uint      total=OrdersTotal();  
//--- пройдем в цикле по всем ордерам  
for(uint i=0;i<total;i++)  
{  
    //--- получим тикет ордера по его позиции в списке  
    if((ticket=OrderGetTicket(i))>0)  
    {  
        //--- получим свойства ордера  
        open_price     =OrderGetDouble(ORDER_PRICE_OPEN);  
        time_setup     =(datetime)OrderGetInteger(ORDER_TIME_SETUP);  
        symbol         =OrderGetString(ORDER_SYMBOL);  
        order_magic    =OrderGetInteger(ORDER_MAGIC);  
        positionID     =OrderGetInteger(ORDER_POSITION_ID);  
        initial_volume=OrderGetDouble(ORDER_VOLUME_INITIAL);  
        type           =EnumToString(ENUM_ORDER_TYPE(OrderGetInteger(ORDER_TYPE)));  
        //--- подготовим и выведем информацию об ордере  
        printf("#ticket %d %s %G %s at %G was set up at %s",  
               ticket,                      // тикет ордера  
               type,                        // тип  
               initial_volume,             // выставленный объем  
               symbol,                     // символ, по которому выставили  
               open_price,                 // указанная цена открытия  
               TimeToString(time_setup)// время установки ордера  
        );  
    }  
}  
//---
```

#### Смотри также

[OrdersTotal\(\)](#), [OrderSelect\(\)](#), [OrderGetInteger\(\)](#)

## OrderSelect

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\(\)](#).

```
bool OrderSelect(
    ulong ticket          // тикет ордера
);
```

### Параметры

*ticket*

[in] Тикет ордера.

### Возвращаемое значение

Значение типа bool.

### Примечание

Не следует путать между собой действующие [отложенные ордера](#) и позиции, которые также отображаются на вкладке "Торговля" в панели "Инструменты" клиентского терминала.

При "неттингом" учте позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций.

Функция OrderSelect() копирует данные об ордере в программное окружение, и последующие вызовы [OrderGetDouble\(\)](#), [OrderGetInteger\(\)](#), [OrderGetString\(\)](#) возвращают ранее скопированные данные. Это означает, что самого ордера может уже и не быть (или же в нем изменились цена открытия, уровни Stop Loss / Take Profit или момент истечения), а данные этому ордеру можно еще получать. Для гарантированного получения свежих данных об ордере рекомендуется вызывать функцию OrderSelect() непосредственно перед обращением за ними.

### Смотри также

[OrderGetInteger\(\)](#), [OrderGetDouble\(\)](#), [OrderGetString\(\)](#), [OrderCalcProfit\(\)](#), [OrderGetTicket\(\)](#),  
[Свойства ордеров](#)

## OrderGetDouble

Возвращает запрошенное свойство ордера, предварительно выбранного при помощи функции [OrderGetTicket](#) или [OrderSelect](#). Свойство ордера должно быть типа double. Существует 2 варианта функции.

- Непосредственно возвращает значение свойства.

```
double OrderGetDouble(
    ENUM_ORDER_PROPERTY_DOUBLE property_id           // идентификатор свойства
);
```

- Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool OrderGetDouble(
    ENUM_ORDER_PROPERTY_DOUBLE property_id,           // идентификатор свойства
    double&                  double_var            // сюда примем значение свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства ордера. Значение может быть одним из значений перечисления [ENUM\\_ORDER\\_PROPERTY\\_DOUBLE](#).

*double\_var*

[out] Переменная типа double, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [double](#). В случае неудачного выполнения возвращает 0.

### Примечание

Не следует путать между собой действующие [отложенные ордера](#) и позиции, которые также отображаются на вкладке "Торговля" в панели "Инструменты" клиентского терминала.

При "неттингом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_NETTING](#)) по каждому символу одновременно может быть открыто несколько позиций.

Для гарантированного получения свежих данных об ордере рекомендуется вызывать функцию [OrderSelect\(\)](#) непосредственно перед обращением за ними.

### Смотри также

[OrdersTotal\(\)](#), [OrderGetTicket\(\)](#), [Свойства ордеров](#)

## OrderGetInteger

Возвращает запрошенное свойство ордера, предварительно выбранного при помощи функции [OrderGetTicket](#) или [OrderSelect](#). Свойство ордера должно быть типа datetime, int. Существует 2 варианта функции.

- Непосредственно возвращает значение свойства.

```
long OrderGetInteger(
    ENUM_ORDER_PROPERTY_INTEGER property_id           // идентификатор свойства
);
```

- Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool OrderGetInteger(
    ENUM_ORDER_PROPERTY_INTEGER property_id,          // идентификатор свойства
    long&           long_var                  // сюда примем значение свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства ордера. Значение может быть одним из значений перечисления [ENUM\\_ORDER\\_PROPERTY\\_INTEGER](#).

*long\_var*

[out] Переменная типа long, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [long](#). В случае неудачного выполнения возвращает 0.

### Примечание

Не следует путать между собой действующие [отложенные ордера](#) и позиции, которые также отображаются на вкладке "Торговля" в панели "Инструменты" клиентского терминала.

При "неттингом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_NETTING](#)) по каждому символу одновременно может быть открыто несколько позиций.

Для гарантированного получения свежих данных об ордере рекомендуется вызывать функцию [OrderSelect\(\)](#) непосредственно перед обращением за ними.

### Смотри также

[OrdersTotal\(\)](#), [OrderGetTicket\(\)](#), [Свойства ордеров](#)

## OrderGetString

Возвращает запрошенное свойство ордера, предварительно выбранного при помощи функции [OrderGetTicket](#) или [OrderSelect](#). Свойство ордера должно быть типа string. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
string OrderGetString(
    ENUM_ORDER_PROPERTY_STRING property_id           // идентификатор свойства
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool OrderGetString(
    ENUM_ORDER_PROPERTY_STRING property_id,          // идентификатор свойства
    string&             string_var                // сюда примем значение свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства ордера. Значение может быть одним из значений перечисления [ENUM\\_ORDER\\_PROPERTY\\_STRING](#).

*string\_var*

[out] Переменная типа string, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [string](#).

### Примечание

Не следует путать между собой действующие [отложенные ордера](#) и позиции, которые также отображаются на вкладке "Торговля" в панели "Инструменты" клиентского терминала.

При "неттингом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_NETTING](#)) по каждому символу одновременно может быть открыто несколько позиций.

Для гарантированного получения свежих данных об ордере рекомендуется вызывать функцию [OrderSelect\(\)](#) непосредственно перед обращением за ними.

### Смотри также

[OrdersTotal\(\)](#), [OrderGetTicket\(\)](#), [Свойства ордеров](#)

## HistorySelect

Запрашивает историю сделок и ордеров за указанный период серверного времени.

```
bool HistorySelect(
    datetime from_date,      // с даты
    datetime to_date         // по дату
);
```

### Параметры

*from\_date*  
 [in] Начальная дата запроса.

*to\_date*  
 [in] Конечная дата запроса.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Функция HistorySelect() создает в mql5-программе список ордеров и список сделок для дальнейшего обращения к элементам списка посредством соответствующих функций. Размер списка сделок можно узнать с помощью функции [HistoryDealsTotal\(\)](#), размер списка ордеров в истории можно получить с [HistoryOrdersTotal\(\)](#). Перебор элементов списка ордеров лучше всего проводить функцией [HistoryOrderGetTicket\(\)](#), для элементов списка сделок соответственно подходит функция [HistoryDealGetTicket\(\)](#).

После применения функции [HistoryOrderSelect\(\)](#) список ордеров в истории, доступных mql5-программе, сбрасывается и заполняется заново найденным ордером, если [поиск ордера по тикету](#) завершился успешно. То же самое относится к списку сделок, доступных mql5-программе - он сбрасывается функцией [HistoryDealSelect\(\)](#) и заполняется заново в случае успешного получения сделки по номеру тикета.

### Пример:

```
void OnStart()
{
    color BuyColor =clrBlue;
    color SellColor=clrRed;
    //--- request trade history
    HistorySelect(0,TimeCurrent());
    //--- create objects
    string name;
    uint total=HistoryDealsTotal();
    ulong ticket=0;
    double price;
    double profit;
    datetime time;
    string symbol;
    long type;
```

```

long      entry;
//--- for all deals
for(uint i=0;i<total;i++)
{
    //--- try to get deals ticket
    if((ticket=HistoryDealGetTicket(i))>0)
    {
        //--- get deals properties
        price =HistoryDealGetDouble(ticket,DEAL_PRICE);
        time  =(datetime)HistoryDealGetInteger(ticket,DEAL_TIME);
        symbol=HistoryDealGetString(ticket,DEAL_SYMBOL);
        type   =HistoryDealGetInteger(ticket,DEAL_TYPE);
        entry  =HistoryDealGetInteger(ticket,DEAL_ENTRY);
        profit=HistoryDealGetDouble(ticket,DEAL_PROFIT);
        //--- only for current symbol
        if(price && time && symbol==Symbol())
        {
            //--- create price object
            name="TradeHistory_Deal_"+string(ticket);
            if(entry) ObjectCreate(0,name,OBJ_ARROW_RIGHT_PRICE,0,time,price,0,0);
            else      ObjectCreate(0,name,OBJ_ARROW_LEFT_PRICE,0,time,price,0,0);
            //--- set object properties
            ObjectSetInteger(0,name,OBJPROP_SELECTABLE,0);
            ObjectSetInteger(0,name,OBJPROP_BACK,0);
            ObjectSetInteger(0,name,OBJPROP_COLOR,type?BuyColor:SellColor);
            if(profit!=0) ObjectSetString(0,name,OBJPROP_TEXT,"Profit: "+string(profit));
        }
    }
}
//--- apply on chart
ChartRedraw();
}

```

#### Смотри также

[HistoryOrderSelect\(\)](#), [HistoryDealSelect\(\)](#)

## HistorySelectByPosition

Запрашивает историю сделок и ордеров, имеющих указанный идентификатор позиции.

```
bool HistorySelectByPosition(
    long position_id      // идентификатор позиции - POSITION IDENTIFIER
);
```

### Параметры

*position\_id*

[in] Идентификатор позиции, который проставляется на каждом исполненном ордере и на каждой сделке.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Не следует путать между собой ордера из торговой истории и действующие [отложенные ордера](#), которые отображаются на вкладке "Торговля" в панели "Инструменты". Список [ордеров](#), которые были отменены или привели к проведению торговой операции, можно посмотреть в закладке "История" на панели "Инструменты" клиентского терминала.

Функция HistorySelectByPosition() создает в mq5-программе список ордеров и список сделок с указанным [идентификатором позиции](#) для дальнейшего обращения к элементам списка посредством соответствующих функций. Размер списка сделок можно узнать с помощью функции [HistoryDealsTotal\(\)](#), размер списка ордеров в истории можно получить с [HistoryOrdersTotal\(\)](#). Перебор элементов списка ордеров лучше всего проводить функцией [HistoryOrderGetTicket\(\)](#), для элементов списка сделок соответственно подходит функция [HistoryDealGetTicket\(\)](#).

После применения функции [HistoryOrderSelect\(\)](#) список ордеров в истории, доступных mq5-программе, сбрасывается и заполняется заново найденным ордером, если [поиск ордера по тикету](#) завершился успешно. То же самое относится к списку сделок, доступных mq5-программе - он сбрасывается функцией [HistoryDealSelect\(\)](#) и заполняется заново в случае успешного получения сделки по номеру тикета.

### Смотри также

[HistorySelect\(\)](#), [HistoryOrderGetTicket\(\)](#), [Свойства ордеров](#)

## HistoryOrderSelect

Выбирает в истории ордер для последующих обращений к нему через соответствующие функции. Возвращает true при успешном завершении функции. Возвращает false при неудачном завершении функции. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

```
bool HistoryOrderSelect(  
    ulong ticket           // тикет ордера  
) ;
```

### Параметры

*ticket*  
[in] Тикет ордера

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Не следует путать между собой ордера из торговой истории и действующие [отложенные ордера](#), которые отображаются на вкладке "Торговля" в панели "Инструменты". Список [ордеров](#), которые были отменены или привели к проведению торговой операции, можно посмотреть в закладке "История" на панели "Инструменты" клиентского терминала.

Функция HistoryOrderSelect() очищает в mq5-программе список ордеров из истории, доступных для обращений, и копирует в него один единственный ордер, если выполнение HistoryOrderSelect() завершилось успешно. Если необходимо перебрать все сделки, выбранные функцией [HistorySelect\(\)](#), то лучше использовать функцию [HistoryOrderGetTicket\(\)](#).

### Смотри также

[HistorySelect\(\)](#), [HistoryOrderGetTicket\(\)](#), [Свойства ордеров](#)

## HistoryOrdersTotal

Возвращает количество ордеров в истории. Перед вызовом функции HistoryOrdersTotal() необходимо получить историю сделок и ордеров с помощью функции [HistorySelect\(\)](#) или [HistorySelectByPosition\(\)](#).

```
int HistoryOrdersTotal();
```

### Возвращаемое значение

Значение типа [int](#).

### Примечание

Не следует путать между собой ордера из торговой истории и действующие [отложенные ордера](#), которые отображаются на вкладке "Торговля" в панели "Инструменты". Список [ордеров](#), которые были отменены или привели к проведению торговой операции, можно посмотреть в закладке "История" на панели "Инструменты" клиентского терминала.

### Смотри также

[HistorySelect\(\)](#), [HistoryOrderSelect\(\)](#), [HistoryOrderGetTicket\(\)](#), [Свойства ордеров](#)

## HistoryOrderGetTicket

Возвращает тикет соответствующего ордера в истории. Перед вызовом функции HistoryOrderGetTicket() необходимо получить историю сделок и ордеров с помощью функции [HistorySelect\(\)](#) или [HistorySelectByPosition\(\)](#).

```
ulong HistoryOrderGetTicket(
    int index // номер в списке ордеров
);
```

### Параметры

*index*  
 [in] Номер ордера в списке ордеров.

### Возвращаемое значение

Значение типа [ulong](#). В случае неудачного выполнения возвращает 0.

### Примечание

Не следует путать между собой ордера из торговой истории и действующие [отложенные ордера](#), которые отображаются на вкладке "Торговля" в панели "Инструменты". Список [ордеров](#), которые были отменены или привели к проведению торговой операции, можно посмотреть в закладке "История" на панели "Инструменты" клиентского терминала.

### Пример:

```
void OnStart()
{
    datetime from=0;
    datetime to=TimeCurrent();
//--- запросить всю историю
    HistorySelect(from,to);
//--- переменные для получения значений из свойств ордера
    ulong ticket;
    double open_price;
    double initial_volume;
    datetime time_setup;
    datetime time_done;
    string symbol;
    string type;
    long order_magic;
    long positionID;
//--- количество текущих отложенных ордеров
    uint total=HistoryOrdersTotal();
//--- пройдем в цикле по всем ордерам
    for(uint i=0;i<total;i++)
    {
        //--- получим тикет ордера по его позиции в списке
        if((ticket=HistoryOrderGetTicket(i))>0)
        {
```

```

//--- получим свойства ордера
open_price= HistoryOrderGetDouble(ticket,ORDER_PRICE_OPEN);
time_setup= (datetime)HistoryOrderGetInteger(ticket,ORDER_TIME_SETUP);
time_done= (datetime)HistoryOrderGetInteger(ticket,ORDER_TIME_DONE);
symbol= HistoryOrderGetString(ticket,ORDER_SYMBOL);
order_magic= HistoryOrderGetInteger(ticket,ORDER_MAGIC);
positionID = HistoryOrderGetInteger(ticket,ORDER_POSITION_ID);
initial_volume= HistoryOrderGetDouble(ticket,ORDER_VOLUME_INITIAL);
type=GetOrderType(HistoryOrderGetInteger(ticket,ORDER_TYPE));
//--- подготовим и выведем информацию об ордере
printf("#ticket %d %s %G %s at %G was set up at %s => done at %s, pos ID=%d",
       ticket,                                     // тикет ордера
       type,                                       // тип
       initial_volume,                            // выставленный объем
       symbol,                                     // символ, по которому выставили
       open_price,                                 // указанная цена открытия
       TimeToString(time_setup), // время установки ордера
       TimeToString(time_done), // время исполнения или удаления
       positionID);                             // ID позиции, в которую влилась сделка по ордеру
);
}
}
//---
}
//+-----+
//| Возвращает строковое наименование типа ордера |
//+-----+
string GetOrderType(long type)
{
  string str_type="unknown operation";
  switch(type)
  {
    case (ORDER_TYPE_BUY): return("buy");
    case (ORDER_TYPE_SELL): return("sell");
    case (ORDER_TYPE_BUY_LIMIT): return("buy limit");
    case (ORDER_TYPE_SELL_LIMIT): return("sell limit");
    case (ORDER_TYPE_BUY_STOP): return("buy stop");
    case (ORDER_TYPE_SELL_STOP): return("sell stop");
    case (ORDER_TYPE_BUY_STOP_LIMIT): return("buy stop limit");
    case (ORDER_TYPE_SELL_STOP_LIMIT): return("sell stop limit");
  }
  return(str_type);
}

```

#### Смотри также

[HistorySelect\(\)](#), [HistoryOrdersTotal\(\)](#), [HistoryOrderSelect\(\)](#), [Свойства ордеров](#)

## HistoryOrderGetDouble

Возвращает запрошенное свойство ордера. Свойство ордера должно быть типа double. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
double HistoryOrderGetDouble(
    ulong             ticket_number,      // тикет
    ENUM_ORDER_PROPERTY_DOUBLE property_id   // идентификатор свойства
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool HistoryOrderGetDouble(
    ulong             ticket_number,      // тикет
    ENUM_ORDER_PROPERTY_DOUBLE property_id, // идентификатор свойства
    double&           double_var        // сюда примем значение свойства
);
```

### Параметры

*ticket\_number*

[in] Тикет ордера.

*property\_id*

[in] Идентификатор свойства ордера. Значение может быть одним из значений перечисления [ENUM\\_ORDER\\_PROPERTY\\_DOUBLE](#).

*double\_var*

[out] Переменная типа double, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [double](#).

### Примечание

Не следует путать между собой ордера из торговой истории и действующие [отложенные ордера](#), которые отображаются на вкладке "Торговля" в панели "Инструменты". Список [ордеров](#), которые были отменены или привели к проведению торговой операции, можно посмотреть в закладке "История" на панели "Инструменты" клиентского терминала.

### Смотри также

[HistorySelect\(\)](#), [HistoryOrdersTotal\(\)](#), [HistoryOrderSelect\(\)](#), [Свойства ордеров](#)

## HistoryOrderGetInteger

Возвращает запрошенное свойство ордера. Свойство ордера должно быть типа datetime, int. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
long HistoryOrderGetInteger(
    ulong                ticket_number,      // тикет
    ENUM_ORDER_PROPERTY_INTEGER property_id     // идентификатор свойства
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool HistoryOrderGetInteger(
    ulong                ticket_number,      // тикет
    ENUM_ORDER_PROPERTY_INTEGER property_id,   // идентификатор свойства
    long&                long_var          // сюда примем значение свойства
);
```

### Параметры

*ticket\_number*

[in] Тикет ордера.

*property\_id*

[in] Идентификатор свойства ордера. Значение может быть одним из значений перечисления [ENUM\\_ORDER\\_PROPERTY\\_INTEGER](#).

*long\_var*

[out] Переменная типа long, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [long](#).

### Примечание

Не следует путать между собой ордера из торговой истории и действующие [отложенные ордера](#), которые отображаются на вкладке "Торговля" в панели "Инструменты". Список [ордеров](#), которые были отменены или привели к проведению торговой операции, можно посмотреть в закладке "История" на панели "Инструменты" клиентского терминала.

### Пример:

```
//+-----+
//| Trade function
//+-----+
void OnTrade()
{
//--- получим тикет последнего ордера из истории торговли за неделю
ulong last_order=GetLastOrderTicket();
```

```

if(HistoryOrderSelect(last_order))
{
    //--- время постановки ордера в миллисекундах от 01.01.1970
    long time_setup_msc=HistoryOrderGetInteger(last_order,ORDER_TIME_SETUP_MSC);
    PrintFormat("Order #%-d ORDER_TIME_SETUP_MSC=%i64 => %s",
               last_order,time_setup_msc,TimeToString(time_setup_msc/1000));
    //--- время исполнения/удаления ордера в миллисекундах от 01.01.1970
    long time_done_msc=HistoryOrderGetInteger(last_order,ORDER_TIME_DONE_MSC);
    PrintFormat("Order #%-d ORDER_TIME_DONE_MSC=%i64 => %s",
               last_order,time_done_msc,TimeToString(time_done_msc/1000));
}

else // сообщим о неудаче
    PrintFormat("HistoryOrderSelect() failed for %-d. Error code=%d",
               last_order,GetLastError());

//---
}

//+-----+
//| Возвращает тикет последнего ордера в истории или -1           |
//+-----+
ulong GetLastOrderTicket()
{
//--- запросим историю за последние 7 дней
if(!GetTradeHistory(7))
{
    //--- сообщим о неудачном вызове и вернем -1
    Print(__FUNCTION__," HistorySelect() вернул false");
    return -1;
}
//---

ulong first_order,last_order,orders=HistoryOrdersTotal();
//--- если ордера есть, начинаем работать с ними
if(orders>0)
{
    Print("Orders = ",orders);
    first_order=HistoryOrderGetTicket(0);
    PrintFormat("first_order = %-d",first_order);
    if(orders>1)
    {
        last_order=HistoryOrderGetTicket((int)orders-1);
        PrintFormat("last_order = %-d",last_order);
        return last_order;
    }
    return first_order;
}
//--- не нашли ни одного ордера, вернем -1
return -1;
}
//+-----+

```

```
//| Запрашивает историю за последние дни и вернет false при неудаче |
//+-----+
bool GetTradeHistory(int days)
{
//--- зададим недельный период времени для запроса торговой истории
datetime to=TimeCurrent();
datetime from=to-days*PeriodSeconds(PERIOD_D1);
ResetLastError();
//--- сделаем запрос и проверим результат
if(!HistorySelect(from,to))
{
    Print(__FUNCTION__," HistorySelect=false. Error code=",GetLastError());
    return false;
}
//--- история получена успешно
return true;
}
```

#### Смотри также

[HistorySelect\(\)](#), [HistoryOrdersTotal\(\)](#), [HistoryOrderSelect\(\)](#), [Свойства ордеров](#)

## HistoryOrderGetString

Возвращает запрошенное свойство ордера. Свойство ордера должно быть типа `string`. Существует 2 варианта функции.

- Непосредственно возвращает значение свойства.

```
string HistoryOrderGetString(
    ulong                ticket_number,      // тикет
    ENUM_ORDER_PROPERTY_STRING property_id      // идентификатор свойства
);
```

- Возвращает `true` или `false` в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool HistoryOrderGetString(
    ulong                ticket_number,      // тикет
    ENUM_ORDER_PROPERTY_STRING property_id,    // идентификатор свойства
    string&              string_var        // сюда примем значение свойства
);
```

### Параметры

`ticket_number`

[in] Тикет ордера.

`property_id`

[in] Идентификатор свойства ордера. Значение может быть одним из значений перечисления [ENUM\\_ORDER\\_PROPERTY\\_STRING](#).

`string_var`

[out] Переменная типа `string`, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [string](#).

### Примечание

Не следует путать между собой ордера из торговой истории и действующие [отложенные ордера](#), которые отображаются на вкладке "Торговля" в панели "Инструменты". Список [ордеров](#), которые были отменены или привели к проведению торговой операции, можно посмотреть в закладке "История" на панели "Инструменты" клиентского терминала.

### Смотри также

[HistorySelect\(\)](#), [HistoryOrdersTotal\(\)](#), [HistoryOrderSelect\(\)](#), [Свойства ордеров](#)

## HistoryDealSelect

Выбирает в истории сделку для дальнейших обращений к ней через соответствующие функции. Возвращает true при успешном завершении функции. Возвращает false при неудачном завершении функции. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

```
bool HistoryDealSelect(  
    ulong ticket // тикет сделки  
) ;
```

### Параметры

*ticket*

[in] Тикет сделки

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Не следует путать между собой [ордера](#), [сделки](#) и [позиции](#). Каждая сделка является результатом исполнения некоего ордера, каждая позиция является итоговым результатом одной или нескольких сделок.

Функция HistoryDealSelect() очищает в mq5-программе список сделок, доступных для обращений, и копирует в него одну единственную сделку, если выполнение HistoryDealSelect() завершилось успешно. Если необходимо перебрать все сделки, выбранные функцией [HistorySelect\(\)](#), то лучше использовать функцию [HistoryDealGetTicket\(\)](#).

### Смотри также

[HistorySelect\(\)](#), [HistoryDealGetTicket\(\)](#), [Свойства сделок](#)

## HistoryDealsTotal

Возвращает количество сделок в истории. Перед вызовом функции HistoryDealsTotal() необходимо получить историю сделок и ордеров с помощью функции [HistorySelect\(\)](#) или [HistorySelectByPosition\(\)](#).

```
int HistoryDealsTotal();
```

### Возвращаемое значение

Значение типа [int](#).

### Примечание

Не следует путать между собой [ордера](#), [сделки](#) и [позиции](#). Каждая сделка является результатом исполнения некоего ордера, каждая позиция является итоговым результатом одной или нескольких сделок.

### Смотри также

[HistorySelect\(\)](#), [HistoryDealGetTicket\(\)](#), [Свойства сделок](#)

## HistoryDealGetTicket

Выбирает сделку для дальнейшей обработки и возвращает тикет сделки в истории. Перед вызовом функции HistoryDealGetTicket() необходимо получить историю сделок и ордеров с помощью функции [HistorySelect\(\)](#) или [HistorySelectByPosition\(\)](#).

```
ulong HistoryDealGetTicket(
    int index // номер сделки
);
```

### Параметры

*index*  
 [in] Номер сделки в списке сделок.

### Возвращаемое значение

Значение типа [ulong](#). В случае неудачного выполнения возвращает 0.

### Примечание

Не следует путать между собой [ордера](#), [сделки](#) и [позиции](#). Каждая сделка является результатом исполнения некоего ордера, каждая позиция является итоговым результатом одной или нескольких сделок.

### Пример:

```
void OnStart()
{
    ulong deal_ticket; // тикет сделки
    ulong order_ticket; // тикет ордера, по которому была совершена сделка
    datetime transaction_time; // время совершения сделки
    long deal_type; // тип торговой операции
    long position_ID; // идентификатор позиции
    string deal_description; // описание операции
    double volume; // объем операции
    string symbol; // по какому символу была сделка
//--- установим начальную и конечную дату для запроса истории сделок
    datetime from_date=0; // с самого начала
    datetime to_date=TimeCurrent(); // по текущий момент
//--- запросим историю сделок в указанном интервале
    HistorySelect(from_date,to_date);
//--- общее количество в списке сделок
    int deals=HistoryDealsTotal();
//--- теперь обрабатаем каждую сделку
    for(int i=0;i<deals;i++)
    {
        deal_ticket= HistoryDealGetTicket(i);
        volume= HistoryDealGetDouble(deal_ticket,DEAL_VOLUME);
        transaction_time=(datetime)HistoryDealGetInteger(deal_ticket,DEAL_TIME);
        order_ticket= HistoryDealGetInteger(deal_ticket,DEAL_ORDER);
        deal_type= HistoryDealGetInteger(deal_ticket,DEAL_TYPE);
```

```

symbol=           HistoryDealGetString(deal_ticket,DEAL_SYMBOL);
position_ID=      HistoryDealGetInteger(deal_ticket,DEAL_POSITION_ID);
deal_description= GetDealDescription(deal_type,volume,symbol,order_ticket);

//--- сделаем красивое форматирование для номера сделки
string print_index=StringFormat("% 3d",i);
//--- выведем информацию по сделке
Print(print_index+": deal #",deal_ticket," at ",transaction_time,deal_description);
}

}

//+-----+
//| Возвращает строковое описание операции |
//+-----+
string GetDealDescription(long deal_type,double volume,string symbol,long ticket,long
{
    string descr;
//---

    switch(deal_type)
    {
        case DEAL_TYPE_BALANCE:             return ("balance");
        case DEAL_TYPE_CREDIT:              return ("credit");
        case DEAL_TYPE_CHARGE:              return ("charge");
        case DEAL_TYPE_CORRECTION:         return ("correction");
        case DEAL_TYPE_BUY:                descr="buy"; break;
        case DEAL_TYPE_SELL:               descr="sell"; break;
        case DEAL_TYPE_BONUS:              return ("bonus");
        case DEAL_TYPE_COMMISSION:         return ("additional commission");
        case DEAL_TYPE_COMMISSION_DAILY:   return ("daily commission");
        case DEAL_TYPE_COMMISSION_MONTHLY: return ("monthly commission");
        case DEAL_TYPE_COMMISSION_AGENT_DAILY: return ("daily agent commission");
        case DEAL_TYPE_COMMISSION_AGENT_MONTHLY: return ("monthly agent commission");
        case DEAL_TYPE_INTEREST:           return ("interest rate");
        case DEAL_TYPE_BUY_CANCELED:       descr="cancelled buy deal"; break;
        case DEAL_TYPE_SELL_CANCELED:      descr="cancelled sell deal"; break;
    }
    descr=StringFormat("%s %G %s (order #%d, position ID %d)",
                       descr, // текущее описание
                       volume, // объем сделки
                       symbol, // инструмент сделки
                       ticket, // тикет ордера, вызвавшего сделку
                       pos_ID // ID позиции, в которой участвовала сделка
                     );
}

return(descr);
}

```

#### Смотри также

[HistorySelect\(\)](#), [HistoryDealsTotal\(\)](#), [HistoryDealSelect\(\)](#), Свойства сделок

## HistoryDealGetDouble

Возвращает запрошенное свойство сделки. Свойство сделки должно быть типа double. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
double HistoryDealGetDouble(
    ulong                ticket_number,      // тикет
    ENUM_DEAL_PROPERTY_DOUBLE property_id       // идентификатор свойства
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool HistoryDealGetDouble(
    ulong                ticket_number,      // тикет
    ENUM_DEAL_PROPERTY_DOUBLE property_id,     // идентификатор свойства
    double&              double_var         // сюда примем значение свойства
);
```

### Параметры

*ticket\_number*

[in] Тикет сделки.

*property\_id*

[in] Идентификатор свойства сделки. Значение может быть одним из значений перечисления [ENUM DEAL PROPERTY DOUBLE](#).

*double\_var*

[out] Переменная типа double, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [double](#).

### Примечание

Не следует путать между собой [ордера](#), [сделки](#) и [позиции](#). Каждая сделка является результатом исполнения некоего ордера, каждая позиция является итоговым результатом одной или нескольких сделок.

### Смотри также

[HistoryDealsTotal\(\)](#), [HistorySelect\(\)](#), [HistoryDealGetTicket\(\)](#), [Свойства сделок](#)

## HistoryDealGetInteger

Возвращает запрошенное свойство сделки. Свойство сделки должно быть типа datetime, int. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
long HistoryDealGetInteger(
    ulong             ticket_number,      // тикет
    ENUM_DEAL_PROPERTY_INTEGER property_id   // идентификатор свойства
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool HistoryDealGetInteger(
    ulong             ticket_number,      // тикет
    ENUM_DEAL_PROPERTY_INTEGER property_id, // идентификатор свойства
    long&             long_var          // сюда примем значение свойства
);
```

### Параметры

*ticket\_number*

[in] Тикет сделки.

*property\_id*

[in] Идентификатор свойства сделки. Значение может быть одним из значений перечисления [ENUM DEAL PROPERTY INTEGER](#).

*long\_var*

[out] Переменная типа long, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [long](#).

### Примечание

Не следует путать между собой [ордера](#), [сделки](#) и [позиции](#). Каждая сделка является результатом исполнения некоего ордера, каждая позиция является итоговым результатом одной или нескольких сделок.

### Пример:

```
//+-----+
//| Trade function
//+-----+
void OnTrade()
{
//--- получим тикет последней сделки из истории торговли за неделю
ulong last_deal=GetLastDealTicket();
```

```

if(HistoryDealSelect(last_deal))
{
    //--- время совершения сделки в миллисекундах от 01.01.1970
    long deal_time_msc=HistoryDealGetInteger(last_deal,DEAL_TIME_MSC);
    PrintFormat("Deal #%-d DEAL_TIME_MSC=%i64 => %s",
                last_deal,deal_time_msc,TimeToString(deal_time_msc/1000));
}
else
    PrintFormat("HistoryDealSelect() failed for #%-d. Error code=%d",
                last_deal,GetLastError());
//---
}

//+-----+
//| Возвращает тикет последней сделки в истории или -1 |
//+-----+
ulong GetLastDealTicket()
{
    //--- запросим историю за последние 7 дней
    if(!GetTradeHistory(7))
    {
        //--- сообщим о неудачном вызове и вернем -1
        Print(__FUNCTION__," HistorySelect() вернул false");
        return -1;
    }
//---

    ulong first_deal,last_deal,deals=HistoryOrdersTotal();
//--- если ордера есть, начинаем работать с ними
    if(deals>0)
    {
        Print("Deals = ",deals);
        first_deal=HistoryDealGetTicket(0);
        PrintFormat("first_deal = %-d",first_deal);
        if(deals>1)
        {
            last_deal=HistoryDealGetTicket((int)deals-1);
            PrintFormat("last_deal = %-d",last_deal);
            return last_deal;
        }
        return first_deal;
    }
//--- не нашли ни одной сделки, вернем -1
    return -1;
}
//+-----+
//| Запрашивает историю за последние дни и вернет false при неудаче |
//+-----+
bool GetTradeHistory(int days)
{
    //--- зададим недельный период времени для запроса торговой истории
}

```

```
datetime to=TimeCurrent();
datetime from=to-days*PeriodSeconds(PERIOD_D1);
ResetLastError();

//--- сделаем запрос и проверим результат
if(!HistorySelect(from,to))
{
    Print(__FUNCTION__," HistorySelect=false. Error code=",GetLastError());
    return false;
}

//--- история получена успешно
return true;
}
```

#### Смотри также

[HistoryDealsTotal\(\)](#), [HistorySelect\(\)](#), [HistoryDealGetTicket\(\)](#), [Свойства сделок](#)

## HistoryDealGetString

Возвращает запрошенное свойство сделки. Свойство сделки должно быть типа `string`. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
string HistoryDealGetString(
    ulong                ticket_number,      // тикет
    ENUM_DEAL_PROPERTY_STRING property_id       // идентификатор свойства
);
```

2. Возвращает `true` или `false` в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool HistoryDealGetString(
    ulong                ticket_number,      // тикет
    ENUM_DEAL_PROPERTY_STRING property_id,     // идентификатор свойства
    string&              string_var        // сюда примем значение свойства
);
```

### Параметры

`ticket_number`

[in] Тикет сделки.

`property_id`

[in] Идентификатор свойства сделки. Значение может быть одним из значений перечисления [ENUM DEAL PROPERTY STRING](#).

`string_var`

[out] Переменная типа `string`, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [string](#).

### Примечание

Не следует путать между собой [ордера](#), [сделки](#) и [позиции](#). Каждая сделка является результатом исполнения некоего ордера, каждая позиция является итоговым результатом одной или нескольких сделок.

### Смотри также

[HistoryDealsTotal\(\)](#), [HistorySelect\(\)](#), [HistoryDealGetTicket\(\)](#), [Свойства сделок](#)

## Управление сигналами

Группа функций, предназначенных для управления торговыми сигналами. Данные функции позволяют:

- получать информацию о торговых сигналах, доступных для копирования,
- прочитать или установить настройки копирования торговых сигналов
- произвести подписку на сигнал и ее отмену средствами языка MQL5.

Функция	Действие
<a href="#">SignalBaseGetDouble</a>	Возвращает значение свойства типа double для выбранного сигнала
<a href="#">SignalBaseGetInteger</a>	Возвращает значение свойства типа integer для выбранного сигнала
<a href="#">SignalBaseGetString</a>	Возвращает значение свойства типа string для выбранного сигнала
<a href="#">SignalBaseSelect</a>	Выбирает для работы сигнал из базы торговых сигналов, доступных в терминале
<a href="#">SignalBaseTotal</a>	Возвращает общее количество сигналов, доступных в терминале
<a href="#">SignallInfoGetDouble</a>	Возвращает из настроек копирования торгового сигнала значение свойства типа double
<a href="#">SignallInfoGetInteger</a>	Возвращает из настроек копирования торгового сигнала значение свойства типа integer
<a href="#">SignallInfoGetString</a>	Возвращает из настроек копирования торгового сигнала значение свойства типа string
<a href="#">SignallInfoSetDouble</a>	Устанавливает в настройках копирования торгового сигнала значение свойства типа double
<a href="#">SignallInfoSetInteger</a>	Устанавливает в настройках копирования торгового сигнала значение свойства типа integer
<a href="#">SignalSubscribe</a>	Производит подписку на копирование торгового сигнала
<a href="#">SignalUnsubscribe</a>	Отменяет подписку на копирование торгового сигнала

## SignalBaseGetDouble

Возвращает значение свойства типа [double](#) для выбранного сигнала.

```
double SignalBaseGetDouble(  
    ENUM_SIGNAL_BASE_DOUBLE     property_id      // идентификатор свойства  
) ;
```

### Параметры

*property\_id*

[in] Идентификатор свойства сигнала. Может быть одним из значений перечисления [ENUM\\_SIGNAL\\_BASE\\_DOUBLE](#).

### Возвращаемое значение

Значение типа [double](#) указанного свойства сигнала.

## SignalBaseGetInteger

Возвращает значение свойства типа [integer](#) для выбранного сигнала.

```
long SignalBaseGetInteger(
    ENUM_SIGNAL_BASE_INTEGER      property_id          // идентификатор свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства сигнала. Может быть одним из значений перечисления [ENUM\\_SIGNAL\\_BASE\\_INTEGER](#).

### Возвращаемое значение

Значение типа [integer](#) указанного свойства сигнала.

## SignalBaseGetString

Возвращает значение свойства типа [string](#) для выбранного сигнала.

```
string SignalBaseGetString(
    ENUM_SIGNAL_BASE_STRING     property_id      // идентификатор свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства сигнала. Может быть одним из значений перечисления [ENUM\\_SIGNAL\\_BASE\\_STRING](#).

### Возвращаемое значение

Значение типа [string](#) указанного свойства сигнала.

## SignalBaseSelect

Выбирает для работы сигнал из базы торговых сигналов, доступных в терминале.

```
bool SignalBaseSelect(
    int     index      // индекс записи сигнала
);
```

### Параметры

*index*

[in] Индекс записи сигнала в базе торговых сигналов.

### Возвращаемое значение

Возвращает true при успешном завершении функции или false в случае ошибки. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Пример:

```
void OnStart()
{
//--- запрашиваем общее количество сигналов в базе
int total=SignalBaseTotal();
//--- цикл по всем сигналам
for(int i=0;i<total;i++)
{
//--- выбираем сигнал для дальнейшей работы
if(SignalBaseSelect(i))
{
//--- получение свойств сигнала
long id =SignalBaseGetInteger(SIGNAL_BASE_ID);           // id сигнала
long pips =SignalBaseGetInteger(SIGNAL_BASE_PIPS);        // результат тор
long subscr=SignalBaseGetInteger(SIGNAL_BASE_SUBSCRIBERS); // количество по
string name =SignalBaseGetString(SIGNAL_BASE_NAME);       // имя сигнала
double price =SignalBaseGetDouble(SIGNAL_BASE_PRICE);     // цена подписки
string curr =SignalBaseGetString(SIGNAL_BASE_CURRENCY);   // валюта сигна
//--- выводим все прибыльные бесплатные сигналы с ненулевым количеством подпи
if(price==0.0 && pips>0 && subscr>0)
PrintFormat("id=%d, name=\"%s\", currency=%s, pips=%d, subscribers=%d",id,
}
else PrintFormat("Ошибка выбора сигнала. Код ошибки=%d",GetLastError());
}
}
```

## SignalBaseTotal

Возвращает общее количество сигналов, доступных в терминале.

```
int SignalBaseTotal();
```

### Возвращаемое значение

Общее количество сигналов, доступных в терминале.

## SignalInfoGetDouble

Возвращает из настроек копирования торгового сигнала значение свойства типа [double](#).

```
double SignalInfoGetDouble(  
    ENUM_SIGNAL_INFO_DOUBLE property_id // идентификатор свойства  
) ;
```

### Параметры

*property\_id*

[in] Идентификатор свойства настроек копирования торгового сигнала. Может быть одним из значений перечисления [ENUM\\_SIGNAL\\_INFO\\_DOUBLE](#).

### Возвращаемое значение

Значение типа [double](#) указанного свойства настроек копирования торгового сигнала.

## SignalInfoGetInteger

Возвращает из настроек копирования торгового сигнала значение свойства типа [integer](#).

```
long SignalInfoGetInteger(
    ENUM_SIGNAL_INFO_INTEGER property_id      // идентификатор свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства настроек копирования торгового сигнала. Может быть одним из значений перечисления [ENUM\\_SIGNAL\\_INFO\\_INTEGER](#).

### Возвращаемое значение

Значение типа [integer](#) указанного свойства настроек копирования торгового сигнала.

## SignalInfoGetString

Возвращает из настроек копирования торгового сигнала значение свойства типа [string](#).

```
string SignalInfoGetString(  
    ENUM_SIGNAL_INFO_STRING     property_id      // идентификатор свойства  
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства настроек копирования торгового сигнала. Значение может быть одним из значений перечисления [ENUM\\_SIGNAL\\_INFO\\_STRING](#).

### Возвращаемое значение

Значение типа [string](#) указанного свойства настроек копирования торгового сигнала.

## SignalInfoSetDouble

Устанавливает в настройках копирования торгового сигнала значение свойства типа [double](#).

```
bool SignalInfoSetDouble(
    ENUM_SIGNAL_INFO_DOUBLE   property_id,      // идентификатор свойства
    double                     value            // значение свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства настроек копирования торгового сигнала. Может быть одним из значений перечисления [ENUM\\_SIGNAL\\_INFO\\_DOUBLE](#).

*value*

[in] Значение свойства настроек копирования торгового сигнала.

### Возвращаемое значение

Возвращает true в случае успешного изменения свойства, иначе возвращает false. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

## SignalInfoSetInteger

Устанавливает в настройках копирования торгового сигнала значение свойства типа `integer`.

```
bool SignalInfoSetInteger(
    ENUM_SIGNAL_INFO_INTEGER property_id,      // идентификатор свойства
    long value                                // значение свойства
);
```

### Параметры

*property\_id*

[in] Идентификатор свойства настроек копирования торгового сигнала. Может быть одним из значений перечисления [ENUM\\_SIGNAL\\_INFO\\_INTEGER](#).

*value*

[in] Значение свойства настроек копирования торгового сигнала.

### Возвращаемое значение

Возвращает `true` в случае успешного изменения свойства, иначе возвращает `false`. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызывать функцию [GetLastError\(\)](#).

## SignalSubscribe

Производит подписку на копирование указанного торгового сигнала.

```
bool SignalSubscribe(  
    long    signal_id      // id сигнала  
);
```

### Параметры

*signal\_id*  
[in] Идентификатор сигнала.

### Возвращаемое значение

Возвращает true в случае успешной подписки на копирование торгового сигнала, иначе возвращает false. Чтобы получить дополнительную информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

## SignalUnsubscribe

Отменяет подписку на копирование торгового сигнала.

```
bool SignalUnsubscribe();
```

### Возвращаемое значение

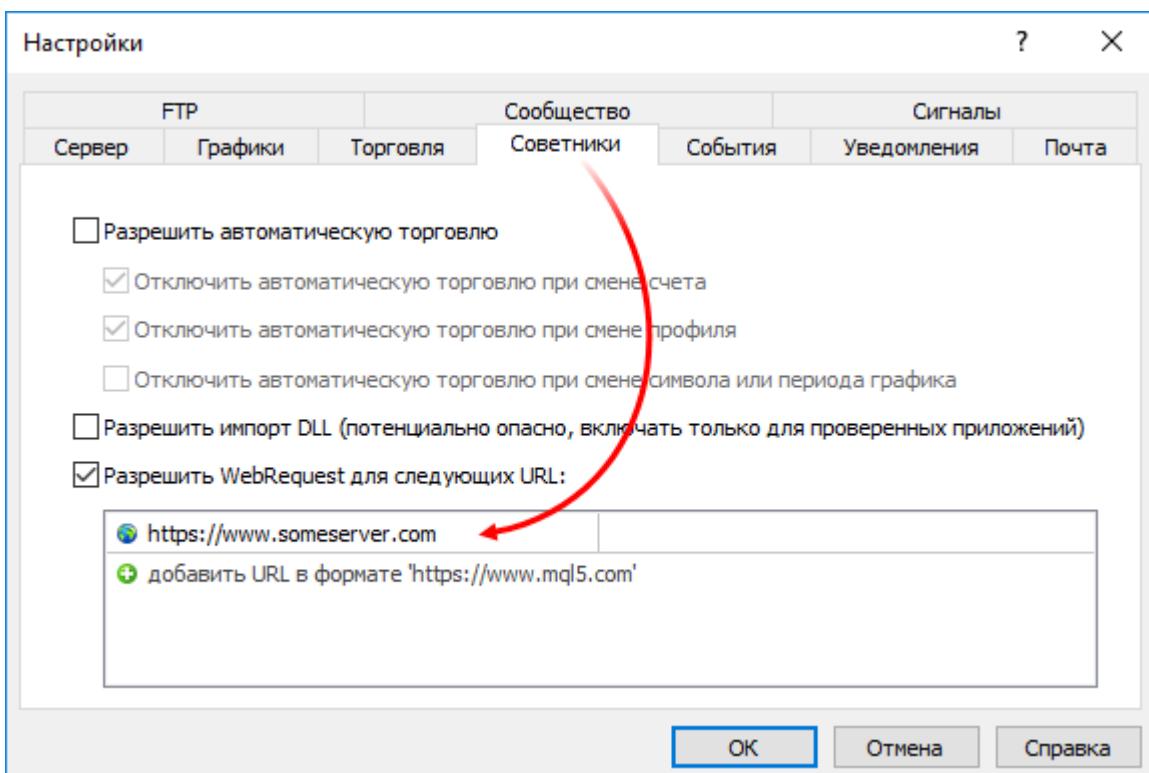
Возвращает true в случае успешной отмены подписки на копирование торгового сигнала, иначе возвращает false. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

## Сетевые функции

Программы MQL5 могут обмениваться данными с удаленными серверами, отправлять push-уведомления, электронные письма и данные по FTP.

- Группа функций [Socket\\*](#) позволяет создать TCP-соединение (в том числе защищенное соединение TLS) с удаленным хостом через системные сокеты. Схема работы проста: вы [создаете сокет](#), [подключаетесь к серверу](#) и можете начинать [чтение](#) и [запись](#) данных.
- Функция [WebRequest](#) предназначена для работы с веб-ресурсами и позволяет легко отправлять HTTP-запросы (в том числе GET и POST).
- [SendFTP](#), [SendMail](#) и [SendNotification](#) – это более простые функции для отправки файлов, электронной почты и мобильных уведомлений.

Для безопасности конечного пользователя на стороне клиентского терминала реализован список разрешенных IP-адресов, с которыми может соединяться MQL5-программа при помощи функций [Socket\\*](#) и [WebRequest](#). Например, если ей требуется подключение к <https://www.someserver.com>, то этот адрес должен быть явно указан пользователем терминала в списке разрешенных. Программно добавить адрес нельзя.



Чтобы уведомить пользователя о необходимости дополнительной настройки, добавьте в MQL5-программу явное сообщение. Например, через [#property description](#), [Alert](#) или [Print](#).

Функция	Действие
<a href="#">SocketCreate</a>	Создает сокет с указанными флагами и возвращает его хэндл
<a href="#">SocketClose</a>	Закрывает сокет

<a href="#"><u>SocketConnect</u></a>	Выполняет подключение к серверу с контролем таймаута
<a href="#"><u>SocketIsConnected</u></a>	Проверяет, подключен ли сокет в текущий момент времени
<a href="#"><u>SocketIsReadable</u></a>	Получает количество байт, которое можно прочитать из сокета
<a href="#"><u>SocketIsWritable</u></a>	Проверяет, возможна ли запись данных в сокет в текущий момент времени
<a href="#"><u>SocketTimeouts</u></a>	Устанавливает таймауты получения и отправки данных для системного объекта сокета
<a href="#"><u>SocketRead</u></a>	Читает данные из сокета
<a href="#"><u>SocketSend</u></a>	Записывает данные в сокете
<a href="#"><u>SocketTlsHandshake</u></a>	Инициирует защищенное TLS (SSL)-соединение с указанным хостом по протоколу TLS Handshake
<a href="#"><u>SocketTlsCertificate</u></a>	Получает данные о сертификате, используемом для защиты сетевого соединения
<a href="#"><u>SocketTlsRead</u></a>	Читает данные из защищенного TLS-соединения
<a href="#"><u>SocketTlsReadAvailable</u></a>	Читает все доступные данные из защищенного TLS-соединения
<a href="#"><u>SocketTlsSend</u></a>	Отправляет данные через защищенное TLS-соединение
<a href="#"><u>WebRequest</u></a>	Отправляет HTTP-запрос на указанный сервер
<a href="#"><u>SendFTP</u></a>	Посыпает файл по адресу, указанному в окне настроек на закладке "FTP"
<a href="#"><u>SendMail</u></a>	Посыпает электронное письмо по адресу, указанному в окне настроек на закладке "Почта"
<a href="#"><u>SendNotification</u></a>	Посыпает Push-уведомления в мобильные терминалы, чьи MetaQuotes ID указаны на закладке "Уведомления"

## SocketCreate

Создает сокет с указанными флагами и возвращает его хэндл.

```
int SocketCreate(
    uint     flags        // флаги
);
```

### Параметры

*flags*

[in] Комбинация флагов, определяющая режим работы с сокетом. В данный момент поддерживается один флаг – SOCKET\_DEFAULT.

### Возвращаемое значение

В случае успешного создания сокета возвращает его хэндл, иначе [INVALID\\_HANDLE](#).

### Примечания

Для освобождения памяти компьютера от неиспользуемого сокета вызовите для него [SocketClose](#).

Из одной MQL5-программы можно создать максимум 128 сокетов. При превышении лимита в [LastError](#) записывается ошибка 5271 (ERR\_NETSOCKET\_TOO\_MANY\_OPENED).

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Пример:

```
//-----+
//|                               SocketExample.mq5  |
//|                               Copyright 2018, MetaQuotes Software Corp.  |
//|                               https://www.mql5.com  |
//+-----+
#property copyright  "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Для работы примера добавьте Address в список разрешенных в наст."
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port     =80;
bool       ExtTLS  =false;
//+-----+
//| Отправка команды на сервер
//+-----+
bool HTTPSend(int socket, string request)
{
    char req[];
    int  len=StringToArray(request,req)-1;
```

```

if(len<0)
    return(false);
//--- если используется защищенное TLS-соединение через порт 443
if(ExtTLS)
    return(SocketTlsSend(socket,req,len)==len);
//--- если используется обычное TCP-соединение
return(SocketSend(socket,req,len)==len);
}

//-----+
//| Чтение ответа сервера |
//-----+
bool HTTPRecv(int socket,uint timeout)
{
    char    rsp[];
    string result;
    uint    timeout_check=GetTickCount()+timeout;
//--- читаем данные из сокета, пока они есть, но не дольше timeout
do
{
    uint len=SocketIsReadable(socket);
    if(len)
    {
        int rsp_len;
        //--- разные команды чтения в зависимости от того, защищенное соединение или
        if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
        else
            rsp_len=SocketRead(socket,rsp,len,timeout);
        //--- разберем ответ
        if(rsp_len>0)
        {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- распечатаем только заголовок ответа
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
            {
                Print("Получен заголовок HTTP ответа:");
                Print(StringSubstr(result,0,header_end));
                return(true);
            }
        }
    }
    while(GetTickCount()<timeout_check && !IsStopped());
    return(false);
}
//-----+
//| Script program start function |
//-----+

```

```

void OnStart()
{
    int socket=SocketCreate();
    //--- проверим хэндл
    if(socket!=INVALID_HANDLE)
    {
        //--- если всё в порядке, подключаемся
        if(SocketConnect(socket,Address,Port,1000))
        {
            Print("Установлено подключение к ",Address,":",Port);

            string subject,issuer,serial,thumbprint;
            datetime expiration;
            //--- если соединение защищено сертификатом, выведем его данные
            if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
            {
                Print("Сертификат TLS:");
                Print("    Владелец: ",subject);
                Print("    Издатель: ",issuer);
                Print("    Номер:     ",serial);
                Print("    Отпечаток: ",thumbprint);
                Print("    Истечение: ",expiration);
                ExtTLS=true;
            }
            //--- отправим на сервер запрос GET
            if(HTTPSnd(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\n\r\n"))
            {
                Print("GET-запрос отправлен");
                //--- прочитаем ответ
                if(!HTTPRcv(socket,1000))
                    Print("Не удалось получить ответ, ошибка ",GetLastError());
            }
            else
                Print("Не удалось отправить GET-запрос, ошибка ",GetLastError());
        }
        else
        {
            Print("Подключение к ",Address,":",Port," не удалось, ошибка ",GetLastError());
        }
        //--- закроем сокет после использования
        SocketClose(socket);
    }
    else
        Print("Не удалось создать сокет, ошибка ",GetLastError());
}
//+-----+

```

## SocketClose

Закрывает сокет.

```
bool SocketClose(
    const int socket      // хэндл сокета
);
```

### Параметры

*socket*

[in] Хэндл сокета, который необходимо закрыть. Хэндл возвращается функцией [SocketCreate](#). При передаче неверного хэндла в [LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Если для сокета ранее было создано соединение через [SocketConnect](#), оно будет разорвано.

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Пример:

```
//+-----+
//|                               SocketExample.mq5  |
//|                               Copyright 2018, MetaQuotes Software Corp.  |
//|                               https://www.mql5.com  |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Для работы примера добавьте Address в список разрешенных в наст."
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port     =80;
bool        ExtTLS  =false;
//+-----+
//| Отправка команды на сервер
//+-----+
bool HTTPSend(int socket,string request)
{
    char req[];
    int len=StringToCharArray(request,req)-1;
    if(len<0)
        return(false);
}
```

```

//--- если используется защищенное TLS-соединение через порт 443
if(ExtTLS)
    return(SocketTlsSend(socket,req,len)==len);
//--- если используется обычное TCP-соединение
return(SocketSend(socket,req,len)==len);
}

//+-----+
//| Чтение ответа сервера |
//+-----+
bool HTTPRecv(int socket,uint timeout)
{
    char    rsp[];
    string result;
    uint    timeout_check=GetTickCount()+timeout;
//--- читаем данные из сокета, пока они есть, но не дольше timeout
do
{
    uint len=SocketIsReadable(socket);
    if(len)
    {
        int rsp_len;
//--- разные команды чтения в зависимости от того, защищенное соединение или
        if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
        else
            rsp_len=SocketRead(socket,rsp,len,timeout);
//--- разберем ответ
        if(rsp_len>0)
        {
            result+=CharArrayToString(rsp,0,rsp_len);
//--- распечатаем только заголовок ответа
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
            {
                Print("Получен заголовок HTTP ответа:");
                Print(StringSubstr(result,0,header_end));
                return(true);
            }
        }
    }
}
while(GetTickCount()<timeout_check && !IsStopped());
return(false);
}

//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
}

```

```

int socket=SocketCreate();
//--- проверим хэндл
if(socket!=INVALID_HANDLE)
{
    //--- если всё в порядке, подключаемся
    if(SocketConnect(socket,Address,Port,1000))
    {
        Print("Установлено подключение к ",Address,":",Port);

        string subject,issuer,serial,thumbprint;
        datetime expiration;
        //--- если соединение защищено сертификатом, выведем его данные
        if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
        {
            Print("Сертификат TLS:");
            Print("    Владелец: ",subject);
            Print("    Издатель: ",issuer);
            Print("    Номер:     ",serial);
            Print("    Отпечаток: ",thumbprint);
            Print("    Истечение: ",expiration);
            ExtTLS=true;
        }
        //--- отправим на сервер запрос GET
        if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\n\r\n"))
        {
            Print("GET-запрос отправлен");
            //--- прочитаем ответ
            if(!HTTPRecv(socket,1000))
                Print("Не удалось получить ответ, ошибка ",GetLastError());
        }
        else
            Print("Не удалось отправить GET-запрос, ошибка ",GetLastError());
    }
    else
    {
        Print("Подключение к ",Address,":",Port," не удалось, ошибка ",GetLastError());
    }
    //--- закроем сокет после использования
    SocketClose(socket);
}
else
    Print("Не удалось создать сокет, ошибка ",GetLastError());
}
//+-----+

```

## SocketConnect

Выполняет подключение к серверу с контролем таймаута.

```
bool SocketConnect(
    int         socket,           // сокет
    const string server,          // адрес для подключения
    uint        port,             // порт для подключения
    uint        timeout_receive_ms // таймаут подключения
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [\\_LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

*server*

[in] Доменное имя сервера, к которому необходимо подключиться, или его IP-адрес.

*port*

[in] Номер порта для подключения.

*timeout\_receive\_ms*

[in] Таймаут подключения в миллисекундах. Если в течение этого времени подключение не удается выполнить, попытки останавливаются.

### Возвращаемое значение

В случае успешного подключения возвращает true, иначе false.

### Примечание

Адрес для подключения должен быть добавлен в список разрешенных на стороне клиентского терминала (раздел Сервис \ Настройки \ Советники).

При ошибке соединения в [\\_LastError](#) записывается ошибка 5272 (ERR\_NETSOCKET\_CANNOT\_CONNECT).

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Пример:

```
//+-----+
//|                               SocketExample.mq5 |
//|                               Copyright 2018, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Для работы примера добавьте Address в список разрешенных в наст." 
```

```

#property script_show_inputs

input string Address="www.mql5.com";
input int Port =80;
bool ExtTLS =false;
//-----
// | Отправка команды на сервер
// +-----+
bool HTTPSend(int socket,string request)
{
    char req[];
    int len=StringToCharArray(request,req)-1;
    if(len<0)
        return(false);
    //--- если используется защищенное TLS-соединение через порт 443
    if(ExtTLS)
        return(SocketTlsSend(socket,req,len)==len);
    //--- если используется обычное TCP-соединение
    return(SocketSend(socket,req,len)==len);
}
//-----
// | Чтение ответа сервера
// +-----+
bool HTTPRecv(int socket,uint timeout)
{
    char rsp[];
    string result;
    uint timeout_check=GetTickCount()+timeout;
    //--- читаем данные из сокета, пока они есть, но не дольше timeout
    do
    {
        uint len=SocketIsReadable(socket);
        if(len)
        {
            int rsp_len;
            //--- разные команды чтения в зависимости от того, защищенное соединение или
            if(ExtTLS)
                rsp_len=SocketTlsRead(socket,rsp,len);
            else
                rsp_len=SocketRead(socket,rsp,len,timeout);
            //--- разберем ответ
            if(rsp_len>0)
            {
                result+=CharArrayToString(rsp,0,rsp_len);
                //--- распечатаем только заголовок ответа
                int header_end=StringFind(result,"\r\n\r\n");
                if(header_end>0)
                {
                    Print("Получен заголовок HTTP ответа:");
                }
            }
        }
    }
}

```

```

        Print(StringSubstr(result,0,header_end));
        return(true);
    }
}
}

while(GetTickCount()<timeout_check && !IsStopped());
return(false);
}

//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
    int socket=SocketCreate();
//--- проверим хэндл
    if(socket!=INVALID_HANDLE)
    {
//--- если всё в порядке, подключаемся
        if(SocketConnect(socket,Address,Port,1000))
        {
            Print("Установлено подключение к ",Address,":",Port);

            string subject,issuer,serial,thumbprint;
            datetime expiration;
//--- если соединение защищено сертификатом, выведем его данные
            if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
            {
                Print("Сертификат TLS:");
                Print("    Владелец: ",subject);
                Print("    Издатель: ",issuer);
                Print("    Номер:     ",serial);
                Print("    Отпечаток: ",thumbprint);
                Print("    Истечение: ",expiration);
                ExtTLS=true;
            }
//--- отправим на сервер запрос GET
            if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\n\r\n"))
            {
                Print("GET-запрос отправлен");
//--- прочитаем ответ
                if(!HTTPRecv(socket,1000))
                    Print("Не удалось получить ответ, ошибка ",GetLastError());
            }
        else
            Print("Не удалось отправить GET-запрос, ошибка ",GetLastError());
    }
}

```

```
Print("Подключение к ",Address,":",Port," не удалось, ошибка ",GetLastError())
}

//--- закроем сокет после использования
SocketClose(socket);
}

else
Print("Не удалось создать сокет, ошибка ",GetLastError());
}

//+-----+
```

## SocketIsConnected

Проверяет, подключен ли сокет в текущий момент времени.

```
bool SocketIsConnected(
    const int socket        // хэндл сокета
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate\(\)](#). При передаче неверного хэндла в [\\_LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

### Возвращаемое значение

Возвращает true, если сокет подключен, иначе false.

### Примечание

С помощью функции `SocketIsConnected()` можно проверить подключение сокета в данный момент времени.

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Смотри также

[SocketConnect](#), [SocketIsWritable](#), [SocketCreate](#), [SocketClose](#)

## SocketIsReadable

Получает количество байт, которое можно прочитать из сокета.

```
uint SocketIsReadable(
    const int socket      // хэндл сокета
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

### Возвращаемое значение

Количество байт, которые можно прочитать. В случае ошибки возвращает 0.

### Примечание

Если при выполнении этой функции на системном сокете произойдет ошибка, соединение, установленное через [SocketConnect](#), будет разорвано.

Перед вызовом [SocketRead](#) проверяйте, есть ли в сокете данные для чтения. В ином случае, при отсутствии данных функция [SocketRead](#) будет впустую ожидать данные в течение `timeout_ms`, задерживая исполнение программы.

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Пример:

```
//+-----+
//|                               SocketExample.mq5  |
//|                               Copyright 2018, MetaQuotes Software Corp.  |
//|                               https://www.mql5.com  |
//+-----+
#property copyright  "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Для работы примера добавьте Address в список разрешенных в наст."
#property script_show_inputs

input string Address="www.mql5.com";
input int     Port     =80;
bool        ExtTLS  =false;
//+-----+
//| Отправка команды на сервер
//+-----+
bool HTTPSend(int socket,string request)
{
    char req[];
```

```

int len=StringToCharArray(request,req)-1;
if(len<0)
    return(false);
//--- если используется защищенное TLS-соединение через порт 443
if(ExtTLS)
    return(SocketTlsSend(socket,req,len)==len);
//--- если используется обычное TCP-соединение
return(SocketSend(socket,req,len)==len);
}

//+-----+
//| Чтение ответа сервера |
//+-----+
bool HTTPRecv(int socket,uint timeout)
{
    char rsp[];
    string result;
    uint timeout_check=GetTickCount()+timeout;
//--- читаем данные из сокета, пока они есть, но не дольше timeout
do
{
    uint len=SocketIsReadable(socket);
    if(len)
    {
        int rsp_len;
        //--- разные команды чтения в зависимости от того, защищенное соединение или
        if(ExtTLS)
            rsp_len=SocketTlsRead(socket,rsp,len);
        else
            rsp_len=SocketRead(socket,rsp,len,timeout);
        //--- разберем ответ
        if(rsp_len>0)
        {
            result+=CharArrayToString(rsp,0,rsp_len);
            //--- распечатаем только заголовок ответа
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
            {
                Print("Получен заголовок HTTP ответа:");
                Print(StringSubstr(result,0,header_end));
                return(true);
            }
        }
    }
    while(GetTickCount()<timeout_check && !IsStopped());
    return(false);
}
//+-----+
//| Script program start function |

```

```

//+-----+
void OnStart()
{
    int socket=SocketCreate();
//--- проверим хэндл
    if(socket!=INVALID_HANDLE)
    {
//--- если всё в порядке, подключаемся
        if(SocketConnect(socket,Address,Port,1000))
        {
            Print("Установлено подключение к ",Address,":",Port);

            string subject,issuer,serial,thumbprint;
            datetime expiration;
//--- если соединение защищено сертификатом, выведем его данные
            if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
            {
                Print("Сертификат TLS:");
                Print("    Владелец: ",subject);
                Print("    Издатель: ",issuer);
                Print("    Номер:      ",serial);
                Print("    Отпечаток: ",thumbprint);
                Print("    Истечение: ",expiration);
                ExtTLS=true;
            }
//--- отправим на сервер запрос GET
            if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\n\r\n"))
            {
                Print("GET-запрос отправлен");
//--- прочитаем ответ
                if(!HTTPRecv(socket,1000))
                    Print("Не удалось получить ответ, ошибка ",GetLastError());
            }
            else
                Print("Не удалось отправить GET-запрос, ошибка ",GetLastError());
        }
    }
    else
    {
        Print("Подключение к ",Address,":",Port," не удалось, ошибка ",GetLastError());
    }
//--- закроем сокет после использования
    SocketClose(socket);
}
else
    Print("Не удалось создать сокет, ошибка ",GetLastError());
//+-----+

```

## SocketIsWritable

Проверяет, возможна ли запись данных в сокет в текущий момент времени.

```
bool SocketIsWritable(
    const int socket           // хэндл сокета
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

### Возвращаемое значение

Возвращает true, если запись возможна, иначе false.

### Примечание

Используя эту функцию, вы можете проверить, возможна ли запись данных в сокет прямо сейчас.

Если при выполнении этой функции на системном сокете произойдет ошибка, соединение, установленное через [SocketConnect](#), будет разорвано.

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

## SocketTimeouts

Устанавливает таймауты получения и отправки данных для системного объекта сокета.

```
bool SocketTimeouts(
    int      socket,           // сокет
    uint     timeout_send_ms,   // таймаут отправки данных
    uint     timeout_receive_ms // таймаут получения данных
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [\\_LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

*timeout\_send\_ms*

[in] Таймаут отправки данных в миллисекундах.

*timeout\_receive\_ms*

[in] Таймаут получения данных в миллисекундах.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Не путайте таймауты системных объектов и таймауты, устанавливаемые при чтении данных через [SocketRead](#). [SocketTimeout](#) устанавливает таймауты один раз для объекта сокета в операционной системе. Эти таймауты будут применяться ко всем функциям чтения и отправки данных через этот сокет. В [SocketRead](#) таймаут устанавливается для конкретной операции чтения данных.

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

## SocketRead

Читает данные из сокета.

```
int SocketRead(
    int         socket,           // сокет
    uchar&     buffer[],        // буфер для чтения данных из сокета
    uint       buffer_maxlen,    // количество байт, которые нужно прочитать
    uint       timeout_ms        // таймаут чтения
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [\\_LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

*buffer*

[out] Ссылка на массив типа [uchar](#), в который будут прочитаны данные. Размер динамического массива увеличивается на количество прочитанных байт. Размер массива не может превышать [INT\\_MAX](#) (2147483647).

*buffer\_maxlen*

[in] Количество байт, которые необходимо прочитать в массив *buffer[]*. Данные, которые не поместятся в массив, останутся в сокете. Их можно будет получить следующим вызовом *SocketRead*. Значение *buffer\_maxlen* не может превышать [INT\\_MAX](#) (2147483647).

*timeout\_ms*

[in] Таймаут чтения данных в миллисекундах. Если в течение этого времени не удаётся получить данные, попытки завершаются и функция возвращает -1.

### Возвращаемое значение

В случае успеха возвращает количество прочитанных байт, в случае ошибки возвращает -1.

### Примечание

Если при выполнении этой функции на системном сокете произойдет ошибка, соединение, установленное через [SocketConnect](#), будет разорвано.

При ошибке чтения данных в [\\_LastError](#) записывается ошибка 5273 (ERR\_NETSOCKET\_IO\_ERROR).

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Пример:

```
//+-----+
//|                               SocketExample.mq5 |
//|                               Copyright 2018, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
```

```

#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Для работы примера добавьте Address в список разрешенных в наст."
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port     =80;
bool        ExtTLS  =false;
//-----
// | Отправка команды на сервер
// +-----+
bool HTTPSend(int socket,string request)
{
    char req[];
    int len=StringToArray(request,req)-1;
    if(len<0)
        return(false);
    //--- если используется защищенное TLS-соединение через порт 443
    if(ExtTLS)
        return(SocketTlsSend(socket,req,len)==len);
    //--- если используется обычное TCP-соединение
    return(SocketSend(socket,req,len)==len);
}
//-----
// | Чтение ответа сервера
// +-----+
bool HTTPRecv(int socket,uint timeout)
{
    char   rsp[];
    string result;
    uint   timeout_check=GetTickCount()+timeout;
    //--- читаем данные из сокета, пока они есть, но не дольше timeout
    do
    {
        uint len=SocketIsReadable(socket);
        if(len)
        {
            int rsp_len;
            //--- разные команды чтения в зависимости от того, защищенное соединение или
            if(ExtTLS)
                rsp_len=SocketTlsRead(socket,rsp,len);
            else
                rsp_len=SocketRead(socket,rsp,len,timeout);
            //--- разберем ответ
            if(rsp_len>0)
            {
                result+=CharArrayToString(rsp,0,rsp_len);
                //--- распечатаем только заголовок ответа
                int header_end=StringFind(result,"\r\n\r\n");
            }
        }
    }
}

```

```

        if(header_end>0)
        {
            Print("Получен заголовок HTTP ответа:");
            Print(StringSubstr(result,0,header_end));
            return(true);
        }
    }
}

while(GetTickCount()<timeout_check && !IsStopped());
return(false);
}

//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
    int socket=SocketCreate();
    //--- проверим хэндл
    if(socket!=INVALID_HANDLE)
    {
        //--- если всё в порядке, подключаемся
        if(SocketConnect(socket,Address,Port,1000))
        {
            Print("Установлено подключение к ",Address,":",Port);

            string subject,issuer,serial,thumbprint;
            datetime expiration;
            //--- если соединение защищено сертификатом, выведем его данные
            if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
            {
                Print("Сертификат TLS:");
                Print("    Владелец: ",subject);
                Print("    Издатель: ",issuer);
                Print("    Номер:     ",serial);
                Print("    Отпечаток: ",thumbprint);
                Print("    Истечение: ",expiration);
                ExtTLS=true;
            }
            //--- отправим на сервер запрос GET
            if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\n\r\n"))
            {
                Print("GET-запрос отправлен");
                //--- прочитаем ответ
                if(!HTTPRecv(socket,1000))
                    Print("Не удалось получить ответ, ошибка ",GetLastError());
            }
        }
        else
            Print("Не удалось отправить GET-запрос, ошибка ",GetLastError());
    }
}

```

```
    }
    else
    {
        Print("Подключение к ",Address,":",Port," не удалось, ошибка ",GetLastError())
    }
//--- закроем сокет после использования
SocketClose(socket);
}
else
{
    Print("Не удалось создать сокет, ошибка ",GetLastError());
}
//+-----+
```

Смотри также

[SocketTimeouts](#), [MathSwap](#)

## SocketSend

Записывает данные в сокете.

```
int SocketSend(
    int         socket,           // сокет
    const uchar& buffer[],       // буфер для данных
    uint        buffer_len        // размер буфера
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [\\_LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

*buffer*

[in] Ссылка на массив типа [uchar](#) с данными, которые необходимо отправить в сокет.

*buffer\_len*

[in] Размер массива buffer.

### Возвращаемое значение

В случае успеха возвращает количество байт, записанных в сокет. В случае ошибки возвращает -1.

### Примечание

Если при выполнении этой функции на системном сокете произойдет ошибка, соединение, установленное через [SocketConnect](#), будет разорвано.

При ошибке записи данных в [\\_LastError](#) записывается ошибка 5273 (ERR\_NETSOCKET\_IO\_ERROR).

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Пример:

```
//+-----+
//|                               SocketExample.mq5 |
//|                               Copyright 2018, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Для работы примера добавьте Address в список разрешенных в наст."
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port     =80;
```

```

bool ExtTLS =false;
//+-----+
//| Отправка команды на сервер |
//+-----+
bool HTTPSend(int socket,string request)
{
    char req[];
    int len=StringToArray(request,req)-1;
    if(len<0)
        return(false);
    //--- если используется защищенное TLS-соединение через порт 443
    if(ExtTLS)
        return(SocketTlsSend(socket,req,len)==len);
    //--- если используется обычное TCP-соединение
    return(SocketSend(socket,req,len)==len);
}
//+-----+
//| Чтение ответа сервера |
//+-----+
bool HTTPRecv(int socket,uint timeout)
{
    char rsp[];
    string result;
    uint timeout_check=GetTickCount()+timeout;
    //--- читаем данные из сокета, пока они есть, но не дольше timeout
    do
    {
        uint len=SocketIsReadable(socket);
        if(len)
        {
            int rsp_len;
            //--- разные команды чтения в зависимости от того, защищенное соединение или
            if(ExtTLS)
                rsp_len=SocketTlsRead(socket,rsp,len);
            else
                rsp_len=SocketRead(socket,rsp,len,timeout);
            //--- разберем ответ
            if(rsp_len>0)
            {
                result+=CharArrayToString(rsp,0,rsp_len);
                //--- распечатаем только заголовок ответа
                int header_end=StringFind(result,"\r\n\r\n");
                if(header_end>0)
                {
                    Print("Получен заголовок HTTP ответа:");
                    Print(StringSubstr(result,0,header_end));
                    return(true);
                }
            }
        }
    }
}

```

```

        }

    }

    while(GetTickCount()<timeout_check && !IsStopped());
    return(false);
}

//+-----+
//| Script program start function
//+-----+

void OnStart()
{
    int socket=SocketCreate();
//--- проверим хэндл
    if(socket!=INVALID_HANDLE)
    {
//--- если всё в порядке, подключаемся
        if(SocketConnect(socket,Address,Port,1000))
        {
            Print("Установлено подключение к ",Address,":",Port);

            string subject,issuer,serial,thumbprint;
            datetime expiration;
//--- если соединение защищено сертификатом, выведем его данные
            if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
            {
                Print("Сертификат TLS:");
                Print("    Владелец: ",subject);
                Print("    Издатель: ",issuer);
                Print("    Номер:     ",serial);
                Print("    Отпечаток: ",thumbprint);
                Print("    Истечение: ",expiration);
                ExtTLS=true;
            }
//--- отправим на сервер запрос GET
            if(HTTPEnd(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\n\r\n"))
            {
                Print("GET-запрос отправлен");
//--- прочитаем ответ
                if(!HTTPRecv(socket,1000))
                    Print("Не удалось получить ответ, ошибка ",GetLastError());
            }
            else
                Print("Не удалось отправить GET-запрос, ошибка ",GetLastError());
        }
        else
            Print("Подключение к ",Address,":",Port," не удалось, ошибка ",GetLastError());
    }
//--- закроем сокет после использования
    SocketClose(socket);
}

```

```
    }
    else
        Print("Не удалось создать сокет, ошибка ", GetLastError());
    }
//+-----+
```

Смотри также

[SocketTimeouts](#), [MathSwap](#), [StringToCharArray](#)

## SocketTlsHandshake

Инициирует защищенное TLS (SSL)-соединение с указанным хостом по протоколу TLS Handshake. Во время Handshake клиент и сервер согласовывают параметры соединения: версию используемого протокола и способ шифрования данных.

```
bool SocketTlsHandshake(
    int          socket,           // сокет
    const string host             // адрес хоста
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

*host*

[in] Адрес хоста, с которым устанавливается защищенное соединение.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечания

До защищенного соединения программа должна установить обычное TCP-соединение с хостом при помощи [SocketConnect](#).

При ошибке установления защищенного соединения в [LastError](#) записывается ошибка 5274 (ERR\_NETSOCKET\_HANDSHAKE\_FAILED).

Вызов этой функции не требуется, если [подключение](#) осуществляется к порту 443. Это стандартный TCP-порт, используемый для защищенных TLS (SSL)-подключений.

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

## SocketTlsCertificate

Получает данные о сертификате, используемом для защиты сетевого соединения.

```
int SocketTlsCertificate(
    int          socket,           // сокет
    string&     subject,          // владелец сертификата
    string&     issuer,           // издатель сертификата
    string&     serial,           // серийный номер сертификата
    string&     thumbprint,        // отпечаток сертификата
    datetime&   expiration        // срок истечения сертификата
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

*subject*

[in] Имя владельца сертификата. Соответствует полю Subject.

*issuer*

[in] Имя издателя сертификата. Соответствует полю Issuer.

*serial*

[in] Серийный номер сертификата. Соответствует полю SerialNumber.

*thumbprint*

[in] Отпечаток сертификата. Соответствует хэшу SHA-1 от всего файла сертификата (все поля, включая подпись издателя).

*expiration*

[in] Срок истечения сертификата в формате [datetime](#).

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Запрос данных от сертификата возможен только после установления защищенного соединения при помощи [SocketTlsHandshake](#).

При ошибке получения сертификата в [LastError](#) записывается ошибка 5275 (ERR\_NETSOCKET\_NO\_CERTIFICATE).

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Пример:

```
//+-----+
```

```

//+-----+
//| Copyright 2018, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Для работы примера добавьте Address в список разрешенных в наст."
#property script_show_inputs

input string Address="www.mql5.com";
input int    Port     =80;
bool       ExtTLS  =false;
//+-----+
//| Отправка команды на сервер
//+-----+
bool HTTPSend(int socket,string request)
{
    char req[];
    int len=StringToArray(request,req)-1;
    if(len<0)
        return(false);
    //--- если используется защищенное TLS-соединение через порт 443
    if(ExtTLS)
        return(SocketTlsSend(socket,req,len)==len);
    //--- если используется обычное TCP-соединение
    return(SocketSend(socket,req,len)==len);
}
//+-----+
//| Чтение ответа сервера
//+-----+
bool HTTPRecv(int socket,uint timeout)
{
    char   rsp[];
    string result;
    uint   timeout_check=GetTickCount()+timeout;
    //--- читаем данные из сокета, пока они есть, но не дольше timeout
    do
    {
        uint len=SocketIsReadable(socket);
        if(len)
        {
            int rsp_len;
            //--- разные команды чтения в зависимости от того, защищенное соединение или
            if(ExtTLS)
                rsp_len=SocketTlsRead(socket,rsp,len);
            else
                rsp_len=SocketRead(socket,rsp,len,timeout);
            //--- разберем ответ
        }
    }
}

```

```

        if(rsp_len>0)
        {
            result+=CharArrayToString(rsp,0, rsp_len);
            //--- распечатаем только заголовок ответа
            int header_end=StringFind(result,"\r\n\r\n");
            if(header_end>0)
            {
                Print("Получен заголовок HTTP ответа:");
                Print(StringSubstr(result,0,header_end));
                return(true);
            }
        }
    }

while(GetTickCount()<timeout_check && !IsStopped());
return(false);
}

//-----+
//| Script program start function
//-----+
void OnStart()
{
    int socket=SocketCreate();
    //--- проверим хэндл
    if(socket!=INVALID_HANDLE)
    {
        //--- если всё в порядке, подключаемся
        if(SocketConnect(socket,Address,Port,1000))
        {
            Print("Установлено подключение к ",Address,":",Port);

            string subject,issuer,serial,thumbprint;
            datetime expiration;
            //--- если соединение защищено сертификатом, выведем его данные
            if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
            {
                Print("Сертификат TLS:");
                Print("    Владелец: ",subject);
                Print("    Издатель: ",issuer);
                Print("    Номер:     ",serial);
                Print("    Отпечаток: ",thumbprint);
                Print("    Истечение: ",expiration);
                ExtTLS=true;
            }
            //--- отправим на сервер запрос GET
            if(HTTPTSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\n\r\n"))
            {
                Print("GET-запрос отправлен");
                //--- прочитаем ответ

```

```
if(!HTTPRecv(socket,1000))
    Print("Не удалось получить ответ, ошибка ",GetLastError());
}
else
    Print("Не удалось отправить GET-запрос, ошибка ",GetLastError());
}
else
{
    Print("Подключение к ",Address,":",Port," не удалось, ошибка ",GetLastError())
}
//--- закроем сокет после использования
SocketClose(socket);
}
else
    Print("Не удалось создать сокет, ошибка ",GetLastError());
}
//+-----+
```

## SocketTlsRead

Читает данные из защищенного TLS-соединения.

```
int SocketTlsRead(
    int          socket,           // сокет
    uchar&      buffer[],        // буфер для чтения данных из сокета
    uint         buffer_maxlen   // количество байт, которые нужно прочитать
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

*buffer*

[out] Ссылка на массив типа [uchar](#), в который будут прочитаны данные. Размер динамического массива увеличивается на количество прочитанных байт. Размер массива не может превышать [INT\\_MAX](#) (2147483647).

*buffer\_maxlen*

[in] Количество байт, которые необходимо прочитать в массив *buffer[]*. Данные, которые не поместятся в массив, останутся в сокете. Их можно будет получить следующим вызовом [SocketTLSRead](#). Значение *buffer\_maxlen* не может превышать [INT\\_MAX](#) (2147483647).

### Возвращаемое значение

В случае успеха возвращает количество прочитанных байт, в случае ошибки возвращает -1.

### Примечание

Если при выполнении этой функции на системном сокете произойдет ошибка, соединение, установленное через [SocketConnect](#), будет разорвано.

Функция исполняется до тех пор, пока не получит указанное количество данных или не наступит таймаут ([SocketTimeouts](#)).

При ошибке чтения данных в [LastError](#) записывается ошибка 5273 (ERR\_NETSOCKET\_IO\_ERROR).

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Пример:

```
//+-----+
//|                               SocketExample.mq5 |
//|                               Copyright 2018, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright  "Copyright 2018, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```

#property description "Для работы примера добавьте Address в список разрешенных в наст."
#property script_show_inputs

input string Address="www.mql5.com";
input int     Port      =80;
bool         ExtTLS   =false;
//+-----+
//| Отправка команды на сервер |
//+-----+
bool HTTPSend(int socket,string request)
{
    char req[];
    int len=StringToArray(request,req)-1;
    if(len<0)
        return(false);
    //--- если используется защищенное TLS-соединение через порт 443
    if(ExtTLS)
        return(SocketTlsSend(socket,req,len)==len);
    //--- если используется обычное TCP-соединение
    return(SocketSend(socket,req,len)==len);
}
//+-----+
//| Чтение ответа сервера |
//+-----+
bool HTTPRecv(int socket,uint timeout)
{
    char   rsp[];
    string result;
    uint   timeout_check=GetTickCount()+timeout;
    //--- читаем данные из сокета, пока они есть, но не дольше timeout
    do
    {
        uint len=SocketIsReadable(socket);
        if(len)
        {
            int rsp_len;
            //--- разные команды чтения в зависимости от того, защищенное соединение или
            if(ExtTLS)
                rsp_len=SocketTlsRead(socket,rsp,len);
            else
                rsp_len=SocketRead(socket,rsp,len,timeout);
            //--- разберем ответ
            if(rsp_len>0)
            {
                result+=CharArrayToString(rsp,0,rsp_len);
                //--- распечатаем только заголовок ответа
                int header_end=StringFind(result,"\r\n\r\n");
                if(header_end>0)
                {

```

```

        Print("Получен заголовок HTTP ответа:");
        Print(StringSubstr(result,0,header_end));
        return(true);
    }
}
}

while(GetTickCount()<timeout_check && !IsStopped());
return(false);
}

//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
    int socket=SocketCreate();
//--- проверим хэндл
    if(socket!=INVALID_HANDLE)
    {
//--- если всё в порядке, подключаемся
        if(SocketConnect(socket,Address,Port,1000))
        {
            Print("Установлено подключение к ",Address,":",Port);

            string subject,issuer,serial,thumbprint;
            datetime expiration;
//--- если соединение защищено сертификатом, выведем его данные
            if(SocketTlsCertificate(socket,subject,issuer,serial,thumbprint,expiration))
            {
                Print("Сертификат TLS:");
                Print("    Владелец: ",subject);
                Print("    Издатель: ",issuer);
                Print("    Номер:     ",serial);
                Print("    Отпечаток: ",thumbprint);
                Print("    Истечение: ",expiration);
                ExtTLS=true;
            }
//--- отправим на сервер запрос GET
            if(HTTPSend(socket,"GET / HTTP/1.1\r\nHost: www.mql5.com\r\n\r\n"))
            {
                Print("GET-запрос отправлен");
//--- прочитаем ответ
                if(!HTTPRecv(socket,1000))
                    Print("Не удалось получить ответ, ошибка ",GetLastError());
            }
        else
            Print("Не удалось отправить GET-запрос, ошибка ",GetLastError());
    }
}

```

```
{  
    Print("Подключение к ",Address,":",Port," не удалось, ошибка ",GetLastError())  
}  
//--- закроем сокет после использования  
SocketClose(socket);  
}  
else  
    Print("Не удалось создать сокет, ошибка ",GetLastError());  
}  
//+-----+
```

Смотри также

[SocketTimeouts](#), [MathSwap](#)

## SocketTlsReadAvailable

Читает все доступные данные из защищенного TLS-соединения.

```
int SocketTlsReadAvailable(
    int          socket,           // сокет
    uchar&      buffer[],        // буфер для чтения данных из сокета
    const uint   buffer_maxlen    // количество байт, которые нужно прочитать
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

*buffer*

[out] Ссылка на массив типа [uchar](#), в который будут прочитаны данные. Размер динамического массива увеличивается на количество прочитанных байт. Размер массива не может превышать [INT\\_MAX](#) (2147483647).

*buffer\_maxlen*

[in] Количество байт, которые необходимо прочитать в массив *buffer[]*. Данные, которые не поместятся в массив, останутся в сокете. Их можно будет получить следующим вызовом [SocketTlsReadAvailable](#) или [SocketTlsRead](#). Значение *buffer\_maxlen* не может превышать [INT\\_MAX](#) (2147483647).

### Возвращаемое значение

В случае успеха возвращает количество прочитанных байт, в случае ошибки возвращает -1.

### Примечание

Если при выполнении этой функции на системном сокете произойдет ошибка, соединение, установленное через [SocketConnect](#), будет разорвано.

При ошибке чтения данных в [LastError](#) записывается ошибка 5273 (ERR\_NETSOCKET\_IO\_ERROR).

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Смотри также

[SocketTimeouts](#), [MathSwap](#)

## SocketTlsSend

Отправляет данные через защищенное TLS-соединение.

```
int SocketTlsSend(
    int         socket,           // сокет
    const uchar& buffer[],       // буфер для данных
    uint        buffer_len        // размер буфера
);
```

### Параметры

*socket*

[in] Хэндл сокета, возвращаемый функцией [SocketCreate](#). При передаче неверного хэндла в [\\_LastError](#) записывается ошибка 5270 (ERR\_NETSOCKET\_INVALIDHANDLE).

*buffer*

[in] Ссылка на массив типа [uchar](#) с данными, которые необходимо отправить.

*buffer\_len*

[in] Размер массива buffer.

### Возвращаемое значение

В случае успеха возвращает количество байт, записанных в сокет. В случае ошибки возвращает -1.

### Примечание

Если при выполнении этой функции на системном сокете произойдет ошибка, соединение, установленное через [SocketConnect](#), будет разорвано.

При ошибке записи данных в [\\_LastError](#) записывается ошибка 5273 (ERR\_NETSOCKET\_IO\_ERROR).

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

### Смотри также

[SocketTimeouts](#), [MathSwap](#), [StringToCharArray](#)

## WebRequest

Отправляет HTTP-запрос на указанный сервер. Существует два варианта функции:

1. Для отправки простых запросов вида "ключ=значение" с использованием заголовка Content-Type: application/x-www-form-urlencoded.

```
int WebRequest(
    const string      method,           // метод HTTP
    const string      url,              // url-адрес
    const string      cookie,           // cookie
    const string      referer,          // referer
    int               timeout,           // таймаут
    const char        &data[],          // массив тела HTTP-сообщения
    int               data_size,         // размер массива data[] в байтах
    char              &result[],         // массив с данными ответа сервера
    string            &result_headers // заголовки ответа сервера
);
```

2. Для отправки запросов произвольного типа с указанием собственного набора заголовков для более гибкого взаимодействия с различными Web-сервисами.

```
int WebRequest(
    const string      method,           // метод HTTP
    const string      url,              // url-адрес
    const string      headers,          // заголовки
    int               timeout,           // таймаут
    const char        &data[],          // массив тела HTTP-сообщения
    char              &result[],         // массив с данными ответа сервера
    string            &result_headers // заголовки ответа сервера
);
```

### Параметры

*method*

[in] Метод HTTP.

*url*

[in] URL-адрес.

*headers*

[in] Заголовки запроса вида "ключ: значение", разделенные переносом строки "\r\n".

*cookie*

[in] Значение Cookie.

*referer*

[in] Значение заголовка Referer HTTP-запроса.

*timeout*

[in] Таймаут в миллисекундах.

```

data[]

[in] Массив данных тела HTTP-сообщения.

data_size

[in] Размер массива data[].

result[]

[out] Массив с данными ответа сервера.

result_headers

[out] Заголовки ответа сервера.

```

### Возвращаемое значение

Код ответа HTTP-сервера либо -1 в случае ошибки.

### Примечание

Для использования функции WebRequest() следует добавить адреса серверов в список разрешенных URL во вкладке "Советники" окна "Настройки". Порт сервера выбирается автоматически на основе указанного протокола - 80 для "http://" и 443 для "https://".

Функция WebRequest() является синхронной, это означает, что она приостанавливает выполнение программы и ждет ответа от запрашиваемого сервера. Так как задержки при получении ответа на отправленный запрос могут быть большими, то функция запрещена для вызовов из индикаторов, поскольку индикаторы работают в едином потоке, общем для всех индикаторов и графиков на данном символе. Задержка выполнения индикатора на одном из графиков символа может привести к остановке обновления всех графиков по данному символу.

Функцию можно вызывать только из экспертов и скриптов, так как они работают в собственном потоке выполнения. При вызове из индикатора [GetLastError\(\)](#) вернет ошибку 4014 - "Системная функция не разрешена для вызова".

При работе в [тестере стратегий](#) функция WebRequest() не выполняется.

### Пример

```

void OnStart()
{
    string cookie=NULL,headers;
    char post[],result[];
    string url="https://finance.yahoo.com";
    //--- для работы с сервером необходимо добавить URL "https://finance.yahoo.com"
    //--- в список разрешенных URL (Главное меню->Сервис->Настройки, вкладка "Советники")
    //--- обнуляем код последней ошибки
    ResetLastError();
    //--- загрузка html-страницы с Yahoo Finance
    int res=WebRequest("GET",url,cookie,NULL,500,post,0,result,headers);
    if(res==-1)
    {
        Print("Ошибка в WebRequest. Код ошибки =",GetLastError());
        //--- возможно, URL отсутствует в списке, выводим сообщение о необходимости его
        MessageBox("Необходимо добавить адрес '"+url+"' в список разрешенных URL во вкла

```

```
        }
    else
    {
        if(res==200)
        {
            //--- успешная загрузка
            PrintFormat("Файл успешно загружен, размер %d байт.",ArraySize(result));
            //PrintFormat("Заголовки сервера: %s",headers);
            //--- сохраняем данные в файл
            int filehandle=FileOpen("url.htm",FILE_WRITE|FILE_BIN);
            if(filehandle!=INVALID_HANDLE)
            {
                //--- сохраняем содержимое массива result[] в файл
                FileWriteArray(filehandle,result,0,ArraySize(result));
                //--- закрываем файл
                FileClose(filehandle);
            }
            else
                Print("Ошибка в FileOpen. Код ошибки =",GetLastError());
        }
        else
            PrintFormat("Ошибка загрузки '%s', код %d",url,res);
    }
}
```

## SendFTP

Посыпает файл по адресу, указанному в окне настроек на закладке "FTP".

```
bool SendFTP(
    string filename,           // файл для отсылки по ftp
    string ftp_path=NULL      // путь для выгрузки на ftp-сервере
);
```

### Параметры

*filename*

[in] Имя отсылаемого файла.

*ftp\_path=NULL*

[in] Каталог FTP. Если каталог не указан, то используется каталог, описанный в настройках.

### Возвращаемое значение

В случае неудачи возвращает `false`.

### Примечание

Отсылаемый файл должен находиться в папке `каталог_терминала\МQL5\files` или ее подпапках. Отсылка не производится, если в настройках не указан адрес FTP и/или пароль доступа.

При работе в [тестере стратегий](#) функция `SendFTP()` не выполняется.

## SendMail

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

```
bool SendMail(
    string subject,           // заголовок
    string some_text           // текст письма
);
```

### Параметры

*subject*

[in] Заголовок письма.

*some\_text*

[in] Тело письма.

### Возвращаемое значение

true - если письмо поставлено в очередь на отсылку, иначе возвращает false.

### Примечание

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

При работе в [тестере стратегий](#) функция SendMail() не выполняется.

## SendNotification

Посыпает уведомление на мобильные терминалы, чьи MetaQuotes ID указаны в окне настроек на закладке "Уведомления".

```
bool SendNotification(  
    string text           // текст сообщения  
);
```

### Параметры

*text*

[in] Текст сообщения в уведомлении. Длина сообщения должна быть не более 255 символов.

### Возвращаемое значение

true при успешной отправке уведомления из терминала, в случае неудачи возвращает false. При проверке после неудачной отправки уведомления [GetLastError\(\)](#) может выдать одну из следующих ошибок:

- 4515 - ERR\_NOTIFICATION\_SEND\_FAILED,
- 4516 - ERR\_NOTIFICATION\_WRONG\_PARAMETER,
- 4517 - ERR\_NOTIFICATION\_WRONG\_SETTINGS,
- 4518 - ERR\_NOTIFICATION\_TOO\_FREQUENT.

### Примечание

Для функции `SendNotification()` установлены жесткие ограничения по использованию: не более 2-х вызовов в секунду и не более 10 вызовов в минуту. Контроль за частотой использования осуществляется динамически, и функция может быть заблокирована при нарушении.

При работе в [тестере стратегий](#) функция `SendNotification()` не выполняется.

## Глобальные переменные клиентского терминала

Группа функций, предназначенных для работы с глобальными переменными.

Не следует путать глобальные переменные клиентского терминала с переменными, объявленными на [глобальном уровне](#) mql5-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 mql5-программ, запущенных на клиентском терминале.

Функция	Действие
<a href="#">GlobalVariableCheck</a>	Проверяет существование глобальной переменной с указанным именем
<a href="#">GlobalVariableTime</a>	Возвращает время последнего доступа к глобальной переменной
<a href="#">GlobalVariableDel</a>	Удаляет глобальную переменную
<a href="#">GlobalVariableGet</a>	Запрашивает значение глобальной переменной
<a href="#">GlobalVariableName</a>	Возвращает имя глобальной переменной по порядковому номеру в списке глобальных переменных
<a href="#">GlobalVariableSet</a>	Устанавливает новое значение глобальной переменной
<a href="#">GlobalVariablesFlush</a>	Принудительно записывает содержимое всех глобальных переменных на диск
<a href="#">GlobalVariableTemp</a>	Устанавливает новое значение глобальной переменной, которая существует только на время текущего сеанса работы терминала
<a href="#">GlobalVariableSetOnCondition</a>	Устанавливает новое значение существующей глобальной переменной по условию
<a href="#">GlobalVariablesDeleteAll</a>	Удаляет глобальные переменные с указанным префиксом в имени
<a href="#">GlobalVariablesTotal</a>	Возвращает общее количество глобальных переменных

## GlobalVariableCheck

Проверяет существование глобальной переменной клиентского терминала.

```
bool GlobalVariableCheck(
    string name // имя
);
```

### Параметры

*name*

[in] Имя глобальной переменной.

### Возвращаемое значение

Возвращает значение `true`, если глобальная переменная существует, иначе возвращает `false`.

### Примечание

Глобальные переменные существуют в клиентском терминале 4 недели с момента последнего обращения, после этого автоматически уничтожаются.

### Смотри также

[GlobalVariableTime\(\)](#)

## GlobalVariableTime

Возвращает время последнего доступа к глобальной переменной.

```
datetime GlobalVariableTime(  
    string name // имя  
) ;
```

### Параметры

*name*

[in] Имя глобальной переменной.

### Возвращаемое значение

Возвращает время последнего доступа к указанной глобальной переменной. Обращение к переменной за значением, например, с помощью функций [GlobalVariableGet\(\)](#) и [GlobalVariableCheck\(\)](#), также изменяет время последнего доступа. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Глобальные переменные существуют в клиентском терминале 4 недели с момента последнего обращения, после этого автоматически уничтожаются.

### Смотри также

[GlobalVariableCheck\(\)](#)

## GlobalVariableDel

Удаляет глобальную переменную клиентского терминала.

```
bool GlobalVariableDel(
    string name          // имя
);
```

### Параметры

*name*

[in] Имя глобальной переменной.

### Возвращаемое значение

При успешном удалении функция возвращает true, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Глобальные переменные существуют в клиентском терминале 4 недели с момента последнего обращения, после этого автоматически уничтожаются.

## GlobalVariableGet

Возвращает значение существующей глобальной переменной клиентского терминала. Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
double GlobalVariableGet(  
    string name          // имя  
) ;
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение глобальной переменной клиентского терминала помещается в приемную переменную, передаваемую по ссылке вторым параметром.

```
bool GlobalVariableGet(  
    string name,           // имя  
    double& double_var     // сюда примем значение глобальной переменной  
) ;
```

### Параметры

*name*

[in] Имя глобальной переменной.

*double\_var*

[out] Переменная типа double, принимающая значение, хранящееся в глобальной переменной клиентского терминала.

### Возвращаемое значение

Значение существующей глобальной переменной или 0 в случае ошибки. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Глобальные переменные существуют в клиентском терминале 4 недели с момента последнего обращения, после этого автоматически уничтожаются.

## GlobalVariableName

Возвращает имя глобальной переменной по порядковому номеру.

```
string GlobalVariableName(
    int index           // номер в списке глобальных переменных
);
```

### Параметры

*index*

[in] Порядковый номер в списке глобальных переменных. Должен быть большим или равным 0 и меньшим, чем [GlobalVariablesTotal\(\)](#).

### Возвращаемое значение

Имя глобальной переменной по порядковому номеру в списке глобальных переменных. Чтобы получить информацию об [ошибке](#) необходимо вызвать функцию функцию [GetLastError\(\)](#).

### Примечание

Глобальные переменные существуют в клиентском терминале 4 недели с момента последнего обращения, после этого автоматически уничтожаются.

## GlobalVariableSet

Устанавливает новое значение глобальной переменной. Если переменная не существует, то система создает новую глобальную переменную.

```
datetime GlobalVariableSet(
    string  name,          // имя
    double  value          // устанавливаемое значение
);
```

### Параметры

*name*

[in] Имя глобальной переменной.

*value*

[in] Новое числовое значение.

### Возвращаемое значение

При успешном выполнении функция возвращает время последнего доступа, иначе 0. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Имя глобальной переменной не должно превышать 63 символов. Глобальные переменные существуют в клиентском терминале 4 недели с момента последнего обращения, после этого автоматически уничтожаются.

## GlobalVariablesFlush

Принудительная запись содержимого всех глобальных переменных на диск.

```
void GlobalVariablesFlush();
```

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

Терминал сам записывает все глобальные переменные при окончании работы, но при внезапном сбое работы компьютера данные могут потеряться. Данная функция позволяет самостоятельно управлять процессом сохранения глобальных переменных на случай непредвиденных ситуаций.

## GlobalVariableTemp

Производит попытку создания временной глобальной переменной. Если переменная не существует, то система создает новую временную глобальную переменную.

```
bool GlobalVariableTemp(  
    string name // имя  
) ;
```

### Параметры

*name*

[in] Имя временной глобальной переменной.

### Возвращаемое значение

При успешном выполнении функция возвращает true, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Временные глобальные переменные существуют только во время работы клиентского терминала, после закрытия терминала они автоматически уничтожаются. При выполнении операции [GlobalVariablesFlush\(\)](#) временные глобальные переменные на диск не записываются.

После создания временной глобальной переменной доступ к ней и ее модификация осуществляется точно так же, как и к обычной [глобальной переменной клиентского терминала](#).

## GlobalVariableSetOnCondition

Устанавливает новое значение существующей глобальной переменной, если текущее значение переменной равно значению третьего параметра `check_value`. Если переменной не существует, функция генерирует ошибку `ERR_GLOBALVARIABLE_NOT_FOUND` (4501) и вернет `false`.

```
bool GlobalVariableSetOnCondition(
    string  name,           // имя
    double  value,          // значение при выполнении условия
    double  check_value     // проверяемое условие
);
```

### Параметры

*name*

[in] Имя глобальной переменной.

*value*

[in] Новое значение.

*check\_value*

[in] Значение для проверки текущего значения глобальной переменной.

### Возвращаемое значение

При успешном выполнении функция возвращает `true`, иначе `false`. Для получения информации об ошибке необходимо вызвать функцию [GetLastError\(\)](#). Если текущее значение глобальной переменной отличается от `check_value`, функция вернет `false`.

### Примечание

Функция обеспечивает атомарный доступ к глобальной переменной, поэтому она может быть использована для организации мьютекса при взаимодействии нескольких одновременно работающих экспертов в пределах одного клиентского терминала.

## GlobalVariablesDeleteAll

Удаляет глобальные переменные клиентского терминала.

```
int GlobalVariablesDeleteAll(
    string      prefix_name=NULL,           // все глобальные переменные, чьи имена начинаются
    datetime    limit_data=0                // все глобальные переменные, которые изменялись с
);
```

### Параметры

*prefix\_name=NULL*

[in] Префикс имени удаляемых глобальных переменных. Если указан префикс NULL либо пустая строка, то под критерий удаления соответствуют все глобальные переменные, соответствующие критерию удаления по дате

*limit\_data=0*

[in] Дата для отбора глобальных переменных по времени последней модификации. Удаляются глобальные переменные, которые изменялись ранее указанной даты. Если параметр равен нулю, то удаляются все глобальные переменные, соответствующие первому критерию (по префиксу).

### Возвращаемое значение

Количество удаленных переменных.

### Примечание

Если оба параметра равны нулю (*prefix\_name=NULL* и *limit\_data=0*), то удаляются все глобальные переменные терминала. Если указаны оба параметра, то удаляются глобальные переменные, соответствующие одновременно каждому из указанных параметров.

Глобальные переменные существуют в клиентском терминале 4 недели с момента последнего обращения, после этого автоматически уничтожаются.

## GlobalVariablesTotal

Возвращает общее количество глобальных переменных клиентского терминала.

```
int GlobalVariablesTotal();
```

### Возвращаемое значение

Количество глобальных переменных.

### Примечание

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.

## Файловые операции

Группа функций для работы с файлами.

Из соображений безопасности в языке MQL5 строго контролируется работа с файлами. Файлы, с которыми проводятся файловые операции средствами языка MQL5, не могут находиться за пределами файловой "песочницы".

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

- папка\_данных\_терминала\MQL5\FILES\ (выберите для просмотра в терминале пункт меню "Файл"- "Открыть каталог данных");
- общая папка всех установленных на компьютере терминалов - обычно расположена в каталоге C:\Documents and Settings\All Users\Application Data\MetaQuotes\Terminal\Common\Files.

Программным путем можно получить наименования этих каталогов с помощью функции [TerminalInfoString\(\)](#), используя перечисление [ENUM\\_TERMINAL\\_INFO\\_STRING](#):

```
//--- Папка, в которой хранятся данные терминала
string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
//--- Общая папка всех клиентских терминалов
string common_data_path=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
```

Работа с файлами из других каталогов пресекается.

Файловые функции позволяют работать с так называемыми "именованными каналами". Для этого достаточно вызвать функцию [FileOpen\(\)](#) с соответствующими параметрами.

Функция	Действие
<a href="#">FileFindFirst</a>	Начинает перебор файлов в соответствующей директории в соответствии с указанным фильтром
<a href="#">FileFindNext</a>	Продолжает поиск, начатый функцией FileFindFirst()
<a href="#">FileFindClose</a>	Закрывает хэндл поиска
<a href="#">FileOpen</a>	Открывает файл с указанным именем и указанными флагами
<a href="#">FileDelete</a>	Удаляет указанный файл
<a href="#">FileFlush</a>	Сброс на диск всех данных, оставшихся в файловом буфере ввода-вывода
<a href="#">FileGetInteger</a>	Получает целочисленное свойство файла
<a href="#">FileIsEnding</a>	Определяет конец файла в процессе чтения
<a href="#">FileIsLineEnding</a>	Определяет конец строки в текстовом файле в процессе чтения
<a href="#">FileClose</a>	Закрывает ранее открытый файл

<a href="#"><u>FileIsExist</u></a>	Проверяет существование файла
<a href="#"><u>FileCopy</u></a>	Копирует исходный файл из локальной или общей папки в другой файл
<a href="#"><u>FileMove</u></a>	Перемещает или переименовывает файл
<a href="#"><u>FileReadArray</u></a>	Читает массивы любых типов, кроме строковых (может быть массив структур, не содержащих строки и динамические массивы), из бинарного файла с текущего положения файлового указателя
<a href="#"><u>FileReadBool</u></a>	Читает из файла типа CSV строку от текущего положения до разделителя (либо до конца текстовой строки) и преобразует прочитанную строку в значение типа bool
<a href="#"><u>FileReadDatetime</u></a>	Читает из файла типа CSV строку одного из форматов: "YYYY.MM.DD HH:MM:SS", "YYYY.MM.DD" или "HH:MM:SS" - и преобразует ее в значение типа datetime
<a href="#"><u>FileReadDouble</u></a>	Читает число двойной точности с плавающей точкой (double) из бинарного файла с текущего положения файлового указателя
<a href="#"><u>FileReadFloat</u></a>	Читает из текущего положения файлового указателя значение типа float
<a href="#"><u>FileReadInteger</u></a>	Читает из бинарного файла значение типа int, short или char в зависимости от указанной длины в байтах
<a href="#"><u>FileReadLong</u></a>	Читает из текущего положения файлового указателя значение типа long
<a href="#"><u>FileReadNumber</u></a>	Читает из файла типа CSV строку от текущего положения до разделителя (либо до конца текстовой строки) и преобразует прочитанную строку в значение типа double
<a href="#"><u>FileReadString</u></a>	Читает из файла строку с текущего положения файлового указателя
<a href="#"><u>FileReadStruct</u></a>	Считывает из бинарного файла содержимое в структуру, переданную в качестве параметра
<a href="#"><u>FileSeek</u></a>	Перемещает положение файлового указателя на указанное количество байт относительно указанного положения
<a href="#"><u>FileSize</u></a>	Возвращает размер соответствующего открытого файла

<a href="#">FileTell</a>	Возвращает текущее положение файлового указателя соответствующего открытого файла
<a href="#">FileWrite</a>	Записывает данные в файл типа CSV или TXT
<a href="#">FileWriteArray</a>	Записывает в файл типа BIN массивы любых типов, кроме строковых
<a href="#">FileWriteDouble</a>	Записывает в бинарный файл значение параметра типа double с текущего положения файлового указателя
<a href="#">FileWriteFloat</a>	Записывает в бинарный файл значение параметра типа float с текущего положения файлового указателя
<a href="#">FileWriteInteger</a>	Записывает в бинарный файл значение параметра типа int с текущего положения файлового указателя
<a href="#">FileWriteLong</a>	Записывает в бинарный файл значение параметра типа long с текущего положения файлового указателя
<a href="#">FileWriteString</a>	Записывает в файл типа BIN или TXT значение параметра типа string с текущего положения файлового указателя
<a href="#">FileWriteStruct</a>	Записывает в бинарный файл содержимое структуры, переданной в качестве параметра, с текущего положения файлового указателя
<a href="#">FileLoad</a>	Считывает всё содержимое указанного бинарного файла в переданный массив числовых типов или простых структур
<a href="#">FileSave</a>	Записывает в бинарный файл все элементы массива, переданного в качестве параметра
<a href="#">FolderCreate</a>	Создает директорию в папке Files (в зависимости от значения common_flag)
<a href="#">FolderDelete</a>	Удаляет указанную директорию. Если папка не пуста, то она не может быть удалена
<a href="#">FolderClean</a>	Удаляет все файлы в указанной папке

Если файл открывается для записи с помощью функции [FileOpen\(\)](#), то все указанные в пути подпапки будут созданы в случае их отсутствия.

## FileFindFirst

Начинает перебор файлов и поддиректорий в соответствующей директории в соответствии с указанным фильтром.

```
long FileFindFirst(
    const string file_filter,           // строка - фильтр поиска
    string& returned_filename,         // имя найденного файла или поддиректории
    int common_flag=0                 // определяет область поиска
);
```

### Параметры

*file\_filter*

[in] Фильтр поиска. В фильтре может быть указана поддиректория (или последовательность вложенных поддиректорий) относительно директории \Files, в которой необходимо проводить перебор файлов.

*returned\_filename*

[out] Возвращаемый параметр, куда в случае удачи помещается имя первого найденного файла или поддиректории. Возвращается только имя файла (включая расширение) без указания директорий и поддиректорий, независимо от того, указывались ли они в фильтре для поиска.

*common\_flag*

[in] Флаг, определяющий местоположение файла. Если common\_flag=FILE\_COMMON, то файл находится в общей папке всех клиентских терминалов \Terminal\Common\Files. В противном случае файл находится в локальной папке.

### Возвращаемое значение

Возвращает хэндл объекта поиска, который необходимо использовать для дальнейшего перебора файлов и поддиректорий функцией [FileFindNext\(\)](#), либо [INVALID\\_HANDLE](#) в случае, когда нет ни одного файла и поддиректории, соответствующего фильтру (в частном случае - директория пуста). После окончания поиска хэндл необходимо закрыть при помощи функции [FileFindClose\(\)](#).

### Примечание

Из соображений безопасности в языке MQL5 строго контролируется работа с файлами. Файлы, с которыми проводятся файловые операции средствами языка MQL5, не могут находиться за пределами файловой "песочницы".

### Пример:

```

//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- фильтр
input string InpFilter="Dir1\\*";
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    string file_name;
    string int_dir="";
    int i=1, pos=0, last_pos=-1;
//--- ищем последний бэк-слеш
    while(!IsStopped())
    {
        pos=StringFind(InpFilter, "\\", pos+1);
        if(pos>=0)
            last_pos=pos;
        else
            break;
    }
//--- в фильтре присутствует имя папки
    if(last_pos>=0)
        int_dir=StringSubstr(InpFilter, 0, last_pos+1);
//--- получение хэндла поиска в корне локальной папки
    long search_handle=FileFindFirst(InpFilter, file_name);
//--- проверим, успешно ли отработала функция FileFindFirst()
    if(search_handle!=INVALID_HANDLE)
    {
//--- в цикле проверим являются ли переданные строки именами файлов или директорий
        do
        {
            ResetLastError();
//--- если это файл, то функция вернет true, а если директория, то функция GetLastError()
            FileIsExist(int_dir+file_name);
            PrintFormat("%d : %s name = %s", i, GetLastError()==ERR_FILE_IS_DIRECTORY ? "D": "F");
            i++;
        }
        while(FileFindNext(search_handle, file_name));
//--- закрываем хэндл поиска
        FileFindClose(search_handle);
    }
    else
        Print("Files not found!");
}

```

#### Смотри также

[FileFindNext](#), [FileFindClose](#)

## FileFindNext

Продолжает поиск, начатый функцией [FileFindFirst\(\)](#).

```
bool FileFindNext(
    long      search_handle,           // handle поиска
    string&   returned_filename     // имя найденного файла или поддиректории
);
```

### Параметры

*search\_handle*

[in] Хэндл поиска, полученный функцией [FileFindFirst\(\)](#).

*returned\_filename*

[out] Имя следующего найденного файла или поддиректории. Возвращается только имя файла (включая расширение) без указания директорий и поддиректорий, независимо от того, указывались ли они в фильтре для поиска.

### Возвращаемое значение

В случае удачи возвращает true, иначе false.

### Пример:

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- фильтр
input string InpFilter="*";
//+-----+
//| Script program start function |+
//+-----+
void OnStart()
{
    string file_name;
    int i=1;
//--- получение хэндла поиска в корне локальной папки
    long search_handle=FileFindFirst(InpFilter,file_name);
//--- проверим, успешно ли отработала функция FileFindFirst()
    if(search_handle!=INVALID_HANDLE)
    {
//--- в цикле проверим являются ли переданные строки именами файлов или директорий
        do
        {
            ResetLastError();
//--- если это файл, то функция вернет true, а если директория, то функция генерирует ошибку
            FileIsExist(file_name);
            PrintFormat("%d : %s name = %s",i,GetLastError()==ERR_FILE_IS_DIRECTORY ? "D" : "F",file_name);
            i++;
        }
        while(FileFindNext(search_handle,file_name));
    }
}
```

```
//--- закрываем хэндл поиска
FileFindClose(search_handle);
}
else
Print("Files not found!");
}
```

Смотри также

[FileFindFirst](#), [FileFindClose](#)

## FileFindClose

Закрывает хэндл поиска.

```
void FileFindClose(
    long search_handle      // handle поиска
);
```

### Параметры

*search\_handle*  
 [in] Хэндл поиска, полученный функцией [FileFindFirst\(\)](#).

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

Функцию необходимо вызывать, чтобы освобождать системные ресурсы.

### Пример:

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- фильтр
input string InpFilter="*";
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
    string file_name;
    int     i=1;
//--- получение хэндла поиска в корне локальной папки
    long search_handle=FileFindFirst(InpFilter,file_name);
//--- проверим, успешно ли отработала функция FileFindFirst()
    if(search_handle!=INVALID_HANDLE)
    {
        //--- в цикле проверим являются ли переданные строки именами файлов или директорий
        do
        {
            ResetLastError();
            //--- если это файл, то функция вернет true, а если директория, то функция генерирует ошибку
            FileIsExist(file_name);
            PrintFormat("%d : %s name = %s",i,GetLastError()==5018 ? "Directory" : "File");
            i++;
        }
        while(FileFindNext(search_handle,file_name));
//--- закрываем хэндл поиска
        FileFindClose(search_handle);
    }
}
```

```
else  
    Print("Files not found!");  
}
```

Смотри также

[FileFindFirst](#), [FileFindNext](#)

## FileIsExist

Проверяет существование файла.

```
bool FileIsExist(
    const string file_name,           // имя файла
    int common_flag=0                // зона поиска
);
```

### Параметры

*file\_name*

[in] Имя проверяемого файла.

*common\_flag=0*

[in] Флаг, определяющий местоположение файла. Если *common\_flag*=FILE\_COMMON, то файл находится в общей папке всех клиентских терминалов \Terminal\Common\Files. В противном случае файл находится в локальной папке.

### Возвращаемое значение

Возвращает true, если указанный файл существует.

### Примечание

Проверяемый файл может оказаться поддиректорией. В этом случае функция FileIsExist() возвратит false, а в переменную \_LastError будет записана ошибка 5018 - "Это не файл, а директория" (смотрите пример для функции [FileFindFirst](#)).

Из соображений безопасности в языке MQL5 строго контролируется работа с файлами. Файлы, с которыми проводятся файловые операции средствами языка MQL5, не могут находиться за пределами файловой "песочницы".

Если *common\_flag*=FILE\_COMMON, то функция ищет указанный файл в общей папке всех клиентских терминалов \Terminal\Common\Files, в противном случае функция ищет файл в локальной папке (MQL5\Files или MQL5\Tester\Files в случае тестирования).

### Пример:

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- дата для старых файлов
input datetime InpFilesDate=D'2013.01.01 00:00';
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    string file_name;           // переменная для хранения имен файлов
    string filter=".txt";       // фильтр для поиска файлов
    datetime create_date;        // дата создания файла
    string files[];             // список имен файлов
    int def_size=25;             // размер массива по умолчанию
```

```

int      size=0;           // количество файлов
//--- выдели память для массива
ArrayResize(files,def_size);
//--- получение хэндла поиска в корне локальной папки
long search_handle=FileFindFirst(filter,file_name);
//--- проверим, успешно ли отработала функция FileFindFirst()
if(search_handle!=INVALID_HANDLE)
{
    //--- в цикле перебираем файлы
    do
    {
        files[size]=file_name;
        //--- увеличим размер массива
        size++;
        if(size==def_size)
        {
            def_size+=25;
            ArrayResize(files,def_size);
        }
        //--- сбрасываем значение ошибки
        ResetLastError();
        //--- получим дату создания файла
        create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,false);
        //--- проверим, старый ли файл
        if(create_date<InpFilesDate)
        {
            PrintFormat("Файл %s удален!",file_name);
            //--- удаляем старый файл
            FileDelete(file_name);
        }
    }
    while(FileFindNext(search_handle,file_name));
    //--- закрываем хэндл поиска
    FileFindClose(search_handle);
}
else
{
    Print("Files not found!");
    return;
}
//--- проверим какие из файлов остались
PrintFormat("Результаты:");
for(int i=0;i<size;i++)
{
    if(FileIsExist(files[i]))
        PrintFormat("Файл %s существует!",files[i]);
    else
        PrintFormat("Файл %s удален!",files[i]);
}

```

{}

**Смотри также**[FileFindFirst](#)

## FileOpen

Функция открывает файл с указанным именем и указанными флагами.

```
int FileOpen(
    string file_name,           // имя файла
    int open_flags,             // комбинация флагов
    short delimiter='\t',        // разделитель
    uint codepage=CP_ACP        // кодовая страница
);
```

### Параметры

*file\_name*

[in] Имя открываемого файла, может содержать подпапки. Если файл открывается для записи, то указанные подпапки будут созданы в случае их отсутствия.

*open\_flags*

[in] комбинация флагов, определяющая режим работы с файлом. Флаги определены следующим образом:

FILE\_READ файл открывается для чтения

FILE\_WRITE файл открывается для записи

FILE\_BIN двоичный режим чтения-записи (без преобразования из строки и в строку)

FILE\_CSV файл типа csv (все записанные элементы преобразуются к строкам соответствующего типа, unicode или ansi, и разделяются разделителем)

FILE\_TXT простой текстовый файл (тот же csv, однако разделитель не принимается во внимание)

FILE\_ANSI строки типа ANSI (однобайтовые символы)

FILE\_UNICODE строки типа UNICODE (двухбайтовые символы)

FILE\_SHARE\_READ совместный доступ по чтению со стороны нескольких программ

FILE\_SHARE\_WRITE совместный доступ по записи со стороны нескольких программ

FILE\_COMMON расположение файла в общей папке всех клиентских терминалов \Terminal\Common\Files.

*delimiter=' \t '*

[in] значение, используемое в качестве разделителя в txt или csv-файле. Если для csv-файла разделитель не указан, то по умолчанию используется символ табуляции. Если для txt-файла разделитель не указан, то никакой разделитель не используется. Если в качестве разделителя явно задано значение 0, то никакой разделитель не используется.

*codepage=CP\_ACP*

[in] Значение кодовой страницы. Для наиболее употребимых кодовых страниц предусмотрены соответствующие константы.

### Возвращаемое значение

В случае успешного открытия функция возвращает хэндл файла, который затем используется для доступа к данным файла. В случае неудачи возвращает INVALID\_HANDLE.

### Примечание

Из соображений безопасности в языке MQL5 строго контролируется работа с файлами. Файлы, с которыми проводятся файловые операции средствами языка MQL5, не могут находиться за пределами файловой "песочницы".

Если файл требуется прочитать в определенной кодировке (указан параметр codepage со значением [кодовой страницы](#)), то необходимо обязательно выставить флаг FILE\_ANSI. Без указания флага FILE\_ANSI чтение текстового файла будет происходить в Юникоде без какого-либо преобразования.

Файл открывается в папке клиентского терминала в подпапке MQL5\Files (или каталог\_агента\_тестирования\MQL5\Files в случае тестирования). Если среди флагов указан FILE\_COMMON, то файл открывается в общей папке всех клиентских терминалов \Terminal\Common\Files.

Можно открывать "именованные каналы" по следующим правилам:

- Имя канала - строка, которая должна иметь вид: "\servername\pipe\pipename", где servername - имя сервера в сети, а pipename - имя канала. Если каналы используются на одном и том же компьютере, имя сервера может быть опущено, но вместо него нужно поставить точку: "\.\pipe\pipename". Клиент, который пытается соединиться с каналом, должен знать его имя.
- Необходимо вызывать [FileFlush\(\)](#) и [FileSeek\(\)](#) на начало файла между последовательными операциями чтения из канала и записи в канал.

В приведенных строках используется специальный символ обратная косая черта '\', поэтому при написании имени в MQL5 программе '\' необходимо удваивать, то есть вышеприведенный пример написать в коде как "\\\servername\\pipe\\pipename".

Более подробно о работе с именованными каналами можно прочитать в статье "[Связь с MetaTrader 5 через именованные каналы без применения DLL](#)"

#### Пример:

```
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- неправильный способ открытия файла
    string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
    string filename=terminal_data_path+"\MQL5\Files\"+"fractals.csv";
    int filehandle=FileOpen(filename,FILE_WRITE|FILE_CSV);
    if(filehandle<0)
    {
        Print("Неудачная попытка открыть файл по абсолютному пути");
        Print("Код ошибки ",GetLastError());
    }
//--- правильный способ работы в "файловой песочнице"
    ResetLastError();
    filehandle=FileOpen("fractals.csv",FILE_WRITE|FILE_CSV);
    if(filehandle!=INVALID_HANDLE)
    {
        FileWrite(filehandle,TimeCurrent(),Symbol(),EnumToString(_Period));
    }
}
```

```
FileClose(filehandle);
Print("FileOpen OK");
}

else Print("Операция FileOpen неудачна, ошибка ", GetLastError());
//--- еще один пример с созданием вложенной директории в MQL5\Files\
string subfolder="Research";
filehandle=FileOpen(subfolder+"\fractals.txt",FILE_WRITE|FILE_CSV);
if(filehandle!=INVALID_HANDLE)
{
    FileWrite(filehandle,TimeCurrent(),Symbol(),EnumToString(_Period));
    FileClose(filehandle);
    Print("Файл должен быть создан в папке "+terminal_data_path+"\\"+subfolder);
}
else Print("Операция FileOpen неудачна, ошибка ", GetLastError());
}
```

#### Смотри также

[Использование кодовой страницы](#), [FileFindFirst](#), [FolderCreate](#), [Флаги открытия файлов](#)

## FileClose

Закрытие файла, ранее открытого функцией [FileOpen\(\)](#).

```
void FileClose(
    int file_handle          // handle файла
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Нет возвращаемого значения.

### Пример:

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input string InpFileName="file.txt";      // имя файла
input string InpDirectoryName="Data";       // имя директории
input int     InpEncodingType=FILE_ANSI;   // ANSI=32 или UNICODE=64
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- распечатаем путь к папке в которой будем работать
PrintFormat("Работаем в папке %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
//--- сбросим значение ошибки
ResetLastError();
//--- откроем файл для чтения (если файл не существует, то произойдет ошибка)
int file_handle=FileOpen(InpDirectoryName+"\\\\"+InpFileName,FILE_READ|FILE_TXT|InpEncodingType);
if(file_handle!=INVALID_HANDLE)
{
//--- распечатаем содержимое файла
while(!FileIsEnding(file_handle))
    Print(FileReadString(file_handle));
//--- закрываем файл
FileClose(file_handle);
}
else
    PrintFormat("Ошибка, код = %d",GetLastError());
}
```

## FileCopy

Копирует исходный файл из локальной или общей папки в другой файл.

```
bool FileCopy(
    const string src_file_name,      // имя файла источника
    int common_flag,                // расположение файла
    const string dst_file_name,      // имя файла назначения
    int mode_flags                  // способ доступа и/или расположение файла
);
```

### Параметры

*src\_file\_name*

[in] Имя файла для копирования.

*common\_flag*

[in] Флаг, определяющий местоположение файла. Если *common\_flag*=FILE\_COMMON, то файл находится в общей папке всех клиентских терминалов \Terminal\Common\Files. В противном случае файл находится в локальной папке (например, *common\_flag*=0).

*dst\_file\_name*

[in] Имя результирующего файла.

*mode\_flags*

[in] Флаги доступа. Параметр может содержать только 2 флага: FILE\_REWRITE и/или FILE\_COMMON - остальные флаги игнорируются. Если файл уже существует и при этом не был указан флаг FILE\_REWRITE, то файл не будет переписан, и функция вернет false.

### Возвращаемое значение

В случае неудачи функция возвращает false.

### Примечание

Из соображений безопасности в языке MQL5 строго контролируется работа с файлами. Файлы, с которыми проводятся файловые операции средствами языка MQL5, не могут находиться за пределами файловой "песочницы".

Если новый файл существовал, то копирование будет производиться в зависимости от наличия флага FILE\_REWRITE в значении параметра mode\_flags.

### Пример:

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input string InpSrc="source.txt";           // источник
input string InpDst="destination.txt";        // копия
input int   InpEncodingType=FILE_ANSI; // ANSI=32 или UNICODE=64
//+-----+
//| Script program start function           |
//+-----+
```

```

void OnStart()
{
//--- отобразим содержимое источника (он должен существовать)
if(!FileDisplay(InpSrc))
    return;
//--- проверим, существует ли уже файл копии (не обязан быть создан)
if(!FileDisplay(InpDst))
{
//--- файл копии не существует, копирование без флага FILE_REWRITE (правильное копирование)
if(FileCopy(InpSrc,0,InpDst,0))
    Print("File is copied!");
else
    Print("File is not copied!");
}
else
{
//--- файл копии уже существует, попробуем скопировать без флага FILE_REWRITE (если копия не удалась, то содержимое файла InpDst останется прежним)
if(FileCopy(InpSrc,0,InpDst,0))
    Print("File is copied!");
else
    Print("File is not copied!");
//--- содержимое файла InpDst останется прежним
FileDisplay(InpDst);
//--- скопируем еще раз с флагом FILE_REWRITE (правильное копирование при существовании файла)
if(FileCopy(InpSrc,0,InpDst,FILE_REWRITE))
    Print("File is copied!");
else
    Print("File is not copied!");
}
//--- получили копию файла InpSrc
FileDisplay(InpDst);
}

//-----+
//| Чтение содержимого файла |
//-----+
bool FileDisplay(const string file_name)
{
//--- сбросим значение ошибки
ResetLastError();
//--- откроем файл
int file_handle=FileOpen(file_name,FILE_READ|FILE_TXT|InpEncodingType);
if(file_handle!=INVALID_HANDLE)
{
//--- отобразим в цикле содержимое файла
Print("-----+");
PrintFormat("File name = %s",file_name);
while(!FileIsEnding(file_handle))
    Print(FileReadString(file_handle));
Print("-----+");
}
}

```

```
//--- закроем файл
FileClose(file_handle);
return(true);
}

//--- не удалось открыть файл
PrintFormat("%s is not opened, error = %d",file_name,GetLastError());
return(false);
}
```

## FileDelete

Удаляет указанный файл в локальной папке клиентского терминала.

```
bool FileDelete(
    const string file_name,      // имя удаляемого файла
    int common_flag=0           // местоположение удаляемого файла
);
```

### Параметры

*file\_name*

[in] Имя файла.

*common\_flag=0*

[in] [Флаг](#), определяющий местоположение файла. Если *common\_flag*=FILE\_COMMON, то файл находится в общей папке всех клиентских терминалов \Terminal\Common\Files. В противном случае файл находится в локальной папке.

### Возвращаемое значение

В случае неудачи функция возвращает `false`.

### Примечание

Из соображений безопасности в языке MQL5 строго контролируется работа с файлами. Файлы, с которыми проводятся файловые операции средствами языка MQL5, не могут находиться за пределами файловой "песочницы".

Удаляет указанный файл в локальной папке клиентского терминала (MQL5\Files или MQL5\Tester\Files в случае тестирования). Если же указан *common\_flag*=FILE\_COMMON, то функция удаляет файл из общей папки всех клиентских терминалов.

### Пример:

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- дата для старых файлов
input datetime InpFilesDate=D'2013.01.01 00:00';
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    string file_name;        // переменная для хранения имен файлов
    string filter=".txt";   // фильтр для поиска файлов
    datetime create_date;    // дата создания файла
    string files[];         // список имен файлов
    int def_size=25;         // размер массива по умолчанию
    int size=0;              // количество файлов
//--- выдели память для массива
    ArrayResize(files,def_size);
//--- получение хэндла поиска в корне локальной папки
```

```
long search_handle=FileFindFirst(filter,file_name);
//--- проверим, успешно ли отработала функция FileFindFirst()
if(search_handle!=INVALID_HANDLE)
{
    //--- в цикле перебираем файлы
    do
    {
        files[size]=file_name;
        //--- увеличим размер массива
        size++;
        if(size==def_size)
        {
            def_size+=25;
            ArrayResize(files,def_size);
        }
        //--- сбрасываем значение ошибки
        ResetLastError();
        //--- получим дату создания файла
        create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,false);
        //--- проверим, старый ли файл
        if(create_date<InpFilesDate)
        {
            PrintFormat("Файл %s удален!",file_name);
            //--- удаляем старый файл
            FileDelete(file_name);
        }
    }
    while(FileFindNext(search_handle,file_name));
    //--- закрываем хэндл поиска
    FileFindClose(search_handle);
}
else
{
    Print("Files not found!");
    return;
}
//--- проверим какие из файлов остались
PrintFormat("Результаты:");
for(int i=0;i<size;i++)
{
    if(FileIsExist(files[i]))
        PrintFormat("Файл %s существует!",files[i]);
    else
        PrintFormat("Файл %s удален!",files[i]);
}
```

## FileMove

Перемещает файл из локальной или общей папки в другую папку.

```
bool FileMove(
    const string src_file_name,           // имя файла для операции перемещения
    int common_flag,                     // расположение файла
    const string dst_file_name,          // имя файла назначения
    int mode_flags                      // способ доступа и/или расположение файла
);
```

### Параметры

*src\_file\_name*

[in] Имя файла для перемещения/переименования.

*common\_flag*

[in] Флаг, определяющий местоположение файла. Если *common\_flag*=FILE\_COMMON, то файл находится в общей папке всех клиентских терминалов \Terminal\Common\Files. В противном случае файл находится в локальной папке. (*common\_flag*=0).

*dst\_file\_name*

[in] Имя результирующего файла.

*mode\_flags*

[in] Флаги доступа. Параметр может содержать только 2 флага: FILE\_REWRITE и/или FILE\_COMMON - остальные флаги игнорируются. Если файл уже существует и при этом не был указан флаг FILE\_REWRITE, то файл не будет переписан, и функция возвратит false.

### Возвращаемое значение

В случае неудачи функция возвращает false.

### Примечание

Из соображений безопасности в языке MQL5 строго контролируется работа с файлами. Файлы, с которыми проводятся файловые операции средствами языка MQL5, не могут находиться за пределами файловой "песочницы".

Если новый файл существовал, то копирование будет производиться в зависимости от наличия флага FILE\_REWRITE в значении параметра mode\_flags.

### Пример:

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input string InpSrcName="data.txt";
input string InpDstName="newdata.txt";
input string InpSrcDirectory="SomeFolder";
input string InpDstDirectory="OtherFolder";
//+-----+
//| Script program start function |
```

```

//+
void OnStart()
{
    string local=TerminalInfoString(TERMINAL_DATA_PATH);
    string common=TerminalInfoString(TERMINAL_COMMANDDATA_PATH);
//--- получим пути к файлам
    string src_path;
    string dst_path;
    StringConcatenate(src_path, InpSrcDirectory, "//", InpSrcName);
    StringConcatenate(dst_path, InpDstDirectory, "//", InpDstName);
//--- проверим, существуют ли файл источника (если нет - выход)
    if(FileIsExist(src_path))
        PrintFormat("%s file exists in the %s\\Files\\%s folder", InpSrcName, local, InpSrcName);
    else
    {
        PrintFormat("Error, %s source file not found", InpSrcName);
        return;
    }
//--- проверим, существует ли уже файл результата
    if(FileIsExist(dst_path,FILE_COMMON))
    {
        PrintFormat("%s file exists in the %s\\Files\\%s folder", InpDstName, common, InpDstName);
//--- файл существует, перемещение нужно проводить с флагом FILE_REWRITE
        ResetLastError();
        if(FileMove(src_path,0,dst_path,FILE_COMMON|FILE_REWRITE))
            PrintFormat("%s file moved", InpSrcName);
        else
            PrintFormat("Error! Code = %d", GetLastError());
    }
    else
    {
        PrintFormat("%s file does not exist in the %s\\Files\\%s folder", InpDstName, common, InpDstName);
//--- файл не существует, перемещение нужно проводить без флага FILE_REWRITE
        ResetLastError();
        if(FileMove(src_path,0,dst_path,FILE_COMMON))
            PrintFormat("%s file moved", InpSrcName);
        else
            PrintFormat("Error! Code = %d", GetLastError());
    }
//--- теперь файл перемещен, проверим это
    if(FileIsExist(dst_path,FILE_COMMON) && !FileIsExist(src_path,0))
        Print("Success!");
    else
        Print("Error!");
}

```

#### Смотри также

[FileIsExist](#)

## FileFlush

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

```
void FileFlush(
    int file_handle // handle файла
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

При выполнении операции записи в файл физически данные могут оказаться в нем только через некоторое время. Для того чтобы данные сразу же сохранились в файле, нужно использовать функцию `FileFlush()`. Если не использовать функцию, то часть данных, еще не попавших на диск, принудительно записывается туда только при закрытии файла функцией `FileClose()`.

Функцию необходимо использовать тогда, когда записываемые данные представляют собой определенную ценность. При этом стоит учитывать, что частый вызов функции может сказаться на скорости работы программы.

Функцию `FileFlush()` необходимо вызывать между операциями чтения из файла и записи в файл.

### Пример:

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- имя файла для записи
input string InpFileName="example.csv"; // имя файла
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- сбросим значение ошибки
    ResetLastError();
//--- откроем файл
    int file_handle=FileOpen(InpFileName,FILE_READ|FILE_WRITE|FILE_CSV);
    if(file_handle!=INVALID_HANDLE)
    {
//--- запишем данные в файл
        for(int i=0;i<1000;i++)
        {
//--- вызовем функцию записи
            FileWrite(file_handle,TimeCurrent(),SymbolInfoDouble(Symbol(),SYMBOL_BID),Sy
//--- сбрасываем данные на диск на каждой 128 итерации
```

```
if((i & 127)==127)
{
    //--- теперь данные будут находиться в файле, и при критической ошибке теряется
    FileFlush(file_handle);
    PrintFormat("i = %d, OK",i);
}
//--- задержка в 0.01 секунды
Sleep(10);
}
//--- закрываем файл
FileClose(file_handle);
}
else
{
    PrintFormat("Ошибка, код = %d",GetLastError());
}
```

Смотри также

[FileClose](#)

## FileGetInteger

Получает целочисленное свойство файла. Существует 2 варианта функции.

### 1. Получение свойств по хэндулу файла.

```
long FileGetInteger(
    int             file_handle,      // хэндл файла
    ENUM_FILE_PROPERTY_INTEGER property_id // идентификатор свойства
);
```

### 2. Получение свойств по имени файла.

```
long FileGetInteger(
    const string      file_name,        // имя файла
    ENUM_FILE_PROPERTY_INTEGER property_id, // идентификатор свойства
    bool             common_folder=false // файл просматривается в локальном каталоге
);
```

#### Параметры

*file\_handle*

[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*file\_name*

[in] Имя файла.

*property\_id*

[in] Идентификатор свойства файла. Значение может быть одним из значений перечисления [ENUM\\_FILE\\_PROPERTY\\_INTEGER](#). Если используется второй вариант функции, то можно получать значения только [следующих свойств](#): FILE\_EXISTS, FILE\_CREATE\_DATE, FILE MODIFY\_DATE, FILE\_ACCESS\_DATE и FILE\_SIZE.

*common\_folder=false*

[in] Указывает на местоположение файла. Если параметр равен false, то просматривается каталог данных терминала, в противном случае предполагается, что файл находится в общей папке всех клиентских терминалов \Terminal\Common\Files ([FILE\\_COMMON](#)).

#### Возвращаемое значение

Значение свойства. В случае ошибки функция возвращает -1, для получения кода ошибки необходимо вызвать функцию [GetLastError\(\)](#).

Если при получении свойств по имени будет указан каталог, то функция в любом случае выставит ошибку 5018 (ERR\_MQL\_FILE\_IS\_DIRECTORY), при этом возвращаемое значение будет корректным.

#### Примечание

Функция всегда изменяет код ошибки. При успешном завершении код ошибки сбрасывается в ноль.

#### Пример:

```

//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input string InpFileName="data.csv";
input string InpDirectoryName="SomeFolder";
//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
    string path=InpDirectoryName+"//"+InpFileName;
    long l=0;
//--- откроем файл
    ResetLastError();
    int handle=FileOpen(path,FILE_READ|FILE_CSV);
    if(handle!=INVALID_HANDLE)
    {
        //--- распечатаем всю информацию о файле
        Print(InpFileName," file info:");
        FileInfo(handle,FILE_EXISTS,l,"bool");
        FileInfo(handle,FILE_CREATE_DATE,l,"date");
        FileInfo(handle,FILE MODIFY_DATE,l,"date");
        FileInfo(handle,FILE_ACCESS_DATE,l,"date");
        FileInfo(handle,FILE_SIZE,l,"other");
        FileInfo(handle,FILE_POSITION,l,"other");
        FileInfo(handle,FILE_END,l,"bool");
        FileInfo(handle,FILE_IS_COMMON,l,"bool");
        FileInfo(handle,FILE_IS_TEXT,l,"bool");
        FileInfo(handle,FILE_IS_BINARY,l,"bool");
        FileInfo(handle,FILE_IS_CSV,l,"bool");
        FileInfo(handle,FILE_IS_ANSI,l,"bool");
        FileInfo(handle,FILE_IS_READABLE,l,"bool");
        FileInfo(handle,FILE_IS_WRITABLE,l,"bool");
        //--- закроем файл
        FileClose(handle);
    }
    else
        PrintFormat("%s file is not opened, ErrorCode = %d",InpFileName,GetLastError());
}
//+-----+
//| Отображение значения свойства файла | 
//+-----+
void FileInfo(const int handle,const ENUM_FILE_PROPERTY_INTEGER id,
              long l,const string type)
{
//--- получим значение свойства
    ResetLastError();
    if((l=FileGetInteger(handle,id))!=-1)
    {

```

```
//--- значение получено, отобразим его в правильном формате
if(!StringCompare(type,"bool"))
    Print(EnumToString(id)," = ",l ? "true" : "false");
if(!StringCompare(type,"date"))
    Print(EnumToString(id)," = ",(datetime)l);
if(!StringCompare(type,"other"))
    Print(EnumToString(id)," = ",l);
}
else
    Print("Error, Code = ",GetLastError());
}
```

#### Смотри также

[Файловые операции](#), [Свойства файлов](#)

## FileIsEnding

Определяет конец файла в процессе чтения.

```
bool FileIsEnding(
    int file_handle // handle файла
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Функция возвращает true в случае, если в процессе чтения или перемещения файлового указателя достигнут конец файла.

### Примечание

Для определения конца файла, функция пытается провести чтение следующей строки из файла. Если ее не существует, то функция возвращает true, в противном случае false.

### Пример:

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input string InpFileName="file.txt"; // имя файла
input string InpDirectoryName="Data"; // имя директории
input int InpEncodingType=FILE_ANSI; // ANSI=32 или UNICODE=64
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- распечатаем путь к папке в которой будем работать
PrintFormat("Работаем в папке %s\\Files\\", TerminalInfoString(TERMINAL_DATA_PATH));
//--- сбросим значение ошибки
ResetLastError();
//--- откроем файл для чтения (если файл не существует, то произойдет ошибка)
int file_handle=FileOpen(InpDirectoryName+"\\\\"+InpFileName, FILE_READ|FILE_TXT|InpEncodingType);
if(file_handle!=INVALID_HANDLE)
{
//--- распечатаем содержимое файла
while(!FileIsEnding(file_handle))
    Print(FileReadString(file_handle));
//--- закрываем файл
FileClose(file_handle);
}
else
    PrintFormat("Ошибка, код = %d", GetLastError());
```

{}

## FileIsLineEnding

Определяет конец строки в текстовом файле в процессе чтения.

```
bool FileIsLineEnding(
    int file_handle          // handle файла
);
```

### Параметры

*file\_handle*

[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Возвращает true в случае, если в процессе чтения txt или csv-файла достигнут конец строки (символы CR-LF).

**Пример** (используется файл, полученный в результате работы примера для функции [FileWriteString](#))

```
//+-----+
//|                               Demo_FileIsLineEnding.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//--- plot Label1
#property indicator_label1 "Overbought & Oversold"
#property indicator_type1 DRAW_COLOR_BARS
#property indicator_color1 clrRed, clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 2
//--- параметры для чтения данных
input string InpFileName="RSI.csv";    // имя файла
input string InpDirectoryName="Data"; // имя директории
//--- индикаторные буферы
double  open_buff[];
double  high_buff[];
double  low_buff[];
double  close_buff[];
double  color_buff[];
//--- переменные перекупленности
int     ovb_ind=0;
int     ovb_size=0;
datetime ovb_time[];
```

```

//--- переменные перепроданности
int      ovs_ind=0;
int      ovs_size=0;
datetime ovs_time[];

//+-----+
//| Custom indicator initialization function           |
//+-----+

int OnInit()
{
//--- переменные размеров массивов по умолчанию
    int ovb_def_size=100;
    int ovs_def_size=100;
//--- выделим память для массивов
    ArrayResize(ovb_time,ovb_def_size);
    ArrayResize(ovs_time,ovs_def_size);
//--- откроем файл
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV|FILE_TEXT);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("Файл %s открыт для чтения",InpFileName);
        PrintFormat("Путь к файлу: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
        double value;
//--- читаем данные из файла
        while(!FileIsEnding(file_handle))
        {
            //--- читаем первое значение в строке
            value=FileReadNumber(file_handle);
            //--- читаем в разные массивы в зависимости от результата функции
            if(value>=70)
                ReadData(file_handle,ovb_time,ovb_size,ovb_def_size);
            else
                ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
        }
//--- закроем файл
        FileClose(file_handle);
        PrintFormat("Данные прочитаны, файл %s закрыт",InpFileName);
    }
    else
    {
        PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
        return(INIT_FAILED);
    }
//--- привязка массивов
    SetIndexBuffer(0,open_buff,INDICATOR_DATA);
    SetIndexBuffer(1,high_buff,INDICATOR_DATA);
    SetIndexBuffer(2,low_buff,INDICATOR_DATA);
    SetIndexBuffer(3,close_buff,INDICATOR_DATA);
    SetIndexBuffer(4,color_buff,INDICATOR_COLOR_INDEX);
}

```

```

//---- установка значений индикатора, которые не будут видимы на графике
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---

return(INIT_SUCCEEDED);
}

//+-----+
//| Чтение данных строки файла |
//+-----+

void ReadData(const int file_handle,datetime &arr[],int &size,int &def_size)
{
    bool flag=false;
//--- читаем пока не дойдем до конца строки или файла
while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
{
    //--- сдвинем каретку, прочитав число
    if(flag)
        FileReadNumber(file_handle);
    //--- запоминаем текущую дату
    arr[size]=FileReadDatetime(file_handle);
    size++;
    //--- при необходимости увеличим размер массива
    if(size==def_size)
    {
        def_size+=100;
        ArrayResize(arr,def_size);
    }
    //--- проскочили первую итерацию
    flag=true;
}
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    ArraySetAsSeries(time,false);
    ArraySetAsSeries(open,false);
    ArraySetAsSeries(high,false);
    ArraySetAsSeries(low,false);
    ArraySetAsSeries(close,false);
}

```

```

//--- цикл для еще необработанных баров
for(int i=prev_calculated;i<rates_total;i++)
{
    //--- по умолчанию 0
    open_buff[i]=0;
    high_buff[i]=0;
    low_buff[i]=0;
    close_buff[i]=0;
    color_buff[i]=0;
    //--- проверка, есть ли еще данные
    if(ovb_ind<ovb_size)
        for(int j=ovb_ind;j<ovb_size;j++)
    {
        //--- если даты совпадают, то бар лежит в зоне перекупленности
        if(time[i]==ovb_time[j])
        {
            open_buff[i]=open[i];
            high_buff[i]=high[i];
            low_buff[i]=low[i];
            close_buff[i]=close[i];
            //--- 0 - красный цвет
            color_buff[i]=0;
            //--- увеличим счетчик
            ovb_ind=j+1;
            break;
        }
    }
    //--- проверка, есть ли еще данные
    if(ovs_ind<ovs_size)
        for(int j=ovs_ind;j<ovs_size;j++)
    {
        //--- если даты совпадают, то бар лежит в зоне перепроданности
        if(time[i]==ovs_time[j])
        {
            open_buff[i]=open[i];
            high_buff[i]=high[i];
            low_buff[i]=low[i];
            close_buff[i]=close[i];
            //--- 1 - синий цвет
            color_buff[i]=1;
            //--- увеличим счетчик
            ovs_ind=j+1;
            break;
        }
    }
}
//--- return value of prev_calculated for next call
return(rates_total);
}

```

```
//+-----+
//| Обработчик события ChartEvent |
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam
)
{
//--- Изменяем толщину индикатора в зависимости от масштаба
if(ChartGetInteger(0,CHART_SCALE)>3)
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,2);
else
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
}
```

#### Смотри также

[FileWriteString](#)

## FileReadArray

Читает массивы любых типов, кроме строковых (может быть массив структур, не содержащих строки и динамические массивы), из бинарного файла с текущего положения файлового указателя.

```
uint FileReadArray(
    int    file_handle,           // handle файла
    void& array[],              // массив для записи
    int    start=0,              // стартовая позиция для записи в массив
    int    count=WHOLE_ARRAY     // сколько читать
);
```

### Параметры

*file\_handle*  
[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*array[]*  
[out] Массив, куда данные будут загружены.

*start=0*  
[in] Стартовая позиция для чтения из массива.

*count=WHOLE\_ARRAY*  
[in] Количество элементов для чтения.

### Возвращаемое значение

Количество прочитанных элементов. По умолчанию, читает весь массив (*count=WHOLE\_ARRAY*).

### Примечание

Строковый массив может читаться только из файла типа ТХТ. При необходимости функция пытается увеличить размер массива.

**Пример** (используется файл, полученный в результате работы примера для функции [FileWriteArray](#))

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+-----+
//| Структура для хранения данных о ценах |
//+-----+
struct prices
{
    datetime      date; // дата
    double        bid; // цена бид
    double        ask; // цена аск
};
```

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- массив структуры
prices arr[];
//--- путь к файлу
string path=InpDirectoryName+"//"+InpFileName;
//--- откроем файл
ResetLastError();
int file_handle=FileOpen(path,FILE_READ|FILE_BIN);
if(file_handle!=INVALID_HANDLE)
{
//--- прочитаем все данные из файла в массив
FileReadArray(file_handle,arr);
//--- получим размер массива
int size=ArraySize(arr);
//--- распечатаем данные из массива
for(int i=0;i<size;i++)
{
Print("Date = ",arr[i].date," Bid = ",arr[i].bid," Ask = ",arr[i].ask);
Print("Total data = ",size);
//--- закрываем файл
FileClose(file_handle);
}
else
Print("File open failed, error ",GetLastError());
}
}
```

#### Смотри также

[Переменные](#), [FileWriteArray](#)

## FileReadBool

Читает из файла типа CSV строку от текущего положения до разделителя (либо до конца текстовой строки) и преобразует прочитанную строку в значение типа bool.

```
bool FileReadBool(
    int file_handle // handle файла
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Прочитанная строка может иметь значения "true", "false" или символьное представление целых чисел "0" или "1". Ненулевое значение преобразуется к логическому true. Функция возвращает полученное преобразованное значение.

**Пример** (используется файл, полученный в результате работы примера для функции [FileWrite\(\)](#))

```
//+-----+
//|                               Demo_FileReadBool.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//---- plot Label1
#property indicator_label1  "UpSignal"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  4
//---- plot Label2
#property indicator_label2  "DownSignal"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  4
//--- параметры для чтения данных
input string InpFileName="MACD.csv"; // имя файла
input string InpDirectoryName="Data"; // имя директории
//--- глобальные переменные
int      ind=0;          // индекс
double  upbuff[];         // индикаторный буфер стрелок вверх
```

```

double    downbuff[]; // индикаторный буфер стрелок вниз
bool      sign_buff[]; // массив сигналов (true - покупка, false - продажа)
datetime  time_buff[]; // массив времени наступления сигналов
int       size=0;      // размер массивов сигналов
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- откроем файл
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV);
if(file_handle!=INVALID_HANDLE)
{
PrintFormat("Файл %s открыт для чтения",InpFileName);
//--- сначала прочитаем количество сигналов
size=(int)FileReadNumber(file_handle);
//--- выделим память для массивов
ArrayResize(sign_buff,size);
ArrayResize(time_buff,size);
//--- прочитаем данные из файла
for(int i=0;i<size;i++)
{
//--- время сигнала
time_buff[i]=FileReadDatetime(file_handle);
//--- значение сигнала
sign_buff[i]=FileReadBool(file_handle);
}
//--- закрываем файл
FileClose(file_handle);
}
else
{
PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
return(INIT_FAILED);
}
//--- привязка массивов
SetIndexBuffer(0,upbuff,INDICATOR_DATA);
SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- зададим код символа для отрисовки в PLOT_ARROW
PlotIndexSetInteger(0,PLOT_ARROW,241);
PlotIndexSetInteger(1,PLOT_ARROW,242);
//--- установка значений индикатора, которые не будут видимы на графике
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
//---
return(INIT_SUCCEEDED);
}
//+-----+

```

```
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    ArraySetAsSeries(time, false);
    ArraySetAsSeries(high, false);
    ArraySetAsSeries(low, false);
    //--- цикл для еще необработанных баров
    for(int i=prev_calculated;i<rates_total;i++)
    {
        //--- по умолчанию 0
        upbuff[i]=0;
        downbuff[i]=0;
        //--- проверка, есть ли еще данные
        if(ind<size)
        {
            for(int j=ind;j<size;j++)
            {
                //--- если даты совпадают, то используем значение из файла
                if(time[i]==time_buff[j])
                {
                    //--- рисуем стрелку в зависимости от сигнала
                    if(sign_buff[j])
                        upbuff[i]=high[i];
                    else
                        downbuff[i]=low[i];
                    //--- увеличим счетчик
                    ind=j+1;
                    break;
                }
            }
        }
    }
    //--- return value of prev_calculated for next call
    return(rates_total);
}
```

#### Смотри также

[Тип bool](#), [FileWrite](#)

## FileReadDatetime

Читает из файла типа CSV строку одного из форматов: "YYYY.MM.DD HH:MI:SS", "YYYY.MM.DD" или "HH:MI:SS" - и преобразует её в значение типа datetime.

```
datetime FileReadDatetime(
    int file_handle // handle файла
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Значение типа datetime.

**Пример** (используется файл, полученный в результате работы примера для функции [FileWrite\(\)](#))

```
//+-----+
//|                               Demo_FileReadDateTIme.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots    2
//---- plot Label1
#property indicator_label1  "UpSignal"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  4
//---- plot Label2
#property indicator_label2  "DownSignal"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  4
//--- параметры для чтения данных
input string InpFileName="MACD.csv"; // имя файла
input string InpDirectoryName="Data"; // имя директории
//--- глобальные переменные
int      ind=0;           // индекс
double   upbuff[];         // индикаторный буфер стрелок вверх
double   downbuff[];        // индикаторный буфер стрелок вниз
bool     sign_buff[];       // массив сигналов (true - покупка, false - продажа)
```

```

datetime time_buff[]; // массив времени наступления сигналов
int size=0; // размер массивов сигналов
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- откроем файл
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("Файл %s открыт для чтения",InpFileName);
//--- сначала прочитаем количество сигналов
size=(int)FileReadNumber(file_handle);
//--- выделим память для массивов
ArrayResize(sign_buff,size);
ArrayResize(time_buff,size);
//--- прочитаем данные из файла
for(int i=0;i<size;i++)
{
    //--- время сигнала
    time_buff[i]=FileReadDatetime(file_handle);
    //--- значение сигнала
    sign_buff[i]=FileReadBool(file_handle);
}
//--- закрываем файл
FileClose(file_handle);
}
else
{
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
    return(INIT_FAILED);
}
//--- привязка массивов
SetIndexBuffer(0,upbuff,INDICATOR_DATA);
SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- зададим код символа для отрисовки в PLOT_ARROW
PlotIndexSetInteger(0,PLOT_ARROW,241);
PlotIndexSetInteger(1,PLOT_ARROW,242);
//--- установка значений индикатора, которые не будут видимы на графике
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+

```

```

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    ArraySetAsSeries(time, false);
    ArraySetAsSeries(high, false);
    ArraySetAsSeries(low, false);

    //--- цикл для еще необработанных баров
    for(int i=prev_calculated;i<rates_total;i++)
    {
        //--- по умолчанию 0
        upbuff[i]=0;
        downbuff[i]=0;
        //--- проверка, есть ли еще данные
        if(ind<size)
        {
            for(int j=ind;j<size;j++)
            {
                //--- если даты совпадают, то используем значение из файла
                if(time[i]==time_buff[j])
                {
                    //--- рисуем стрелку в зависимости от сигнала
                    if(sign_buff[j])
                        upbuff[i]=high[i];
                    else
                        downbuff[i]=low[i];
                    //--- увеличим счетчик
                    ind=j+1;
                    break;
                }
            }
        }
    }
    //--- return value of prev_calculated for next call
    return(rates_total);
}

```

#### Смотри также

[Тип datetime](#), [StringToTime](#), [TimeToString](#), [FileWrite](#)

## FileReadDouble

Читает число двойной точности с плавающей точкой (double) из бинарного файла с текущего положения файлового указателя.

```
double FileReadDouble(
    int file_handle // handle файла
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Значение типа double.

### Примечание

Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

**Пример** (используется файл, полученный в результате работы примера для функции [FileWriteDouble\(\)](#))

```
//+-----+
//|                               Demo_FileReadDouble.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Label1
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_separate_window
//--- параметры для чтения данных
input string InpFileName="MA.csv";      // имя файла
input string InpDirectoryName="Data"; // имя директории
//--- глобальные переменные
int      ind=0;
int      size=0;
double   ma_buff[];
datetime time_buff[];
//--- indicator buffer
```

```

double    buff[];
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- откроем файл
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
if(file_handle!=INVALID_HANDLE)
{
PrintFormat("Файл %s открыт для чтения",InpFileName);
PrintFormat("Путь к файлу: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
//--- сначала прочитаем сколько всего в файле данных
size=(int)FileReadDouble(file_handle);
//--- выделим память для массивов
ArrayResize(ma_buff,size);
ArrayResize(time_buff,size);
//--- прочитаем данные из файла
for(int i=0;i<size;i++)
{
time_buff[i]=(datetime)FileReadDouble(file_handle);
ma_buff[i]=FileReadDouble(file_handle);
}
//--- закроем файл
FileClose(file_handle);
PrintFormat("Данные прочитаны, файл %s закрыт",InpFileName);
}
else
{
PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
return(INIT_FAILED);
}
//--- привязка массива к индикаторному буферу с индексом 0
SetIndexBuffer(0,buff,INDICATOR_DATA);
//--- установка значений индикатора, которые не будут видимы на графике
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function               |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
```

```
        const double &close[],  
        const long &tick_volume[],  
        const long &volume[],  
        const int &spread[])  
{  
    ArraySetAsSeries(time, false);  
    //--- цикл для еще необработанных баров  
    for(int i=prev_calculated;i<rates_total;i++)  
    {  
        //--- по умолчанию 0  
        buff[i]=0;  
        //--- проверка, есть ли еще данные  
        if(ind<size)  
        {  
            for(int j=ind;j<size;j++)  
            {  
                //--- если даты совпадают, то используем значение из файла  
                if(time[i]==time_buff[j])  
                {  
                    buff[i]=ma_buff[j];  
                    //--- увеличим счетчик  
                    ind=j+1;  
                    break;  
                }  
            }  
        }  
    }  
    //--- return value of prev_calculated for next call  
    return(rates_total);  
}
```

#### Смотри также

[Вещественные типы \(double, float\)](#), [StringToDouble](#), [DoubleToString](#), [FileWriteDouble](#)

## FileReadFloat

Читает число одинарной точности с плавающей точкой (float) из текущей позиции бинарного файла.

```
float FileReadFloat(
    int file_handle // handle файла
);
```

### Параметры

`file_handle`  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Значение типа float.

### Примечание

Чтобы получить информацию об ошибке, необходимо вызывать функцию [GetLastError\(\)](#).

**Пример** (используется файл, полученный в результате работы примера для функции [FileWriteFloat\(\)](#))

```
//-----+
//|                               Demo_FileReadFloat.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot Label1
#property indicator_label1 "CloseLine"
#property indicator_type1  DRAW_COLOR_LINE
#property indicator_color1 clrRed,clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 2
//--- параметры для чтения данных
input string InpFileName="Close.bin"; // имя файла
input string InpDirectoryName="Data"; // имя директории
//--- глобальные переменные
int      ind=0;
int      size=0;
double  close_buff[];
datetime time_buff[];
//--- indicator buffers
double  buff[];
```

```

double    color_buff[];
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
    int def_size=100;
//--- выделим память для массивов
    ArrayResize(close_buff,def_size);
    ArrayResize(time_buff,def_size);
//--- откроем файл
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("Файл %s открыт для чтения",InpFileName);
        PrintFormat("Путь к файлу: %s\\Files\\\",TerminalInfoString(TERMINAL_DATA_PATH));
//--- прочитаем данные из файла
        while(!FileIsEnding(file_handle))
        {
            //--- считываем значение времени и цены
            time_buff[size]=(datetime)FileReadDouble(file_handle);
            close_buff[size]=(double)FileReadFloat(file_handle);
            size++;
            //--- увеличим размеры массивов если они переполнены
            if(size==def_size)
            {
                def_size+=100;
                ArrayResize(close_buff,def_size);
                ArrayResize(time_buff,def_size);
            }
        }
//--- закроем файл
        FileClose(file_handle);
        PrintFormat("Данные прочитаны, файл %s закрыт",InpFileName);
    }
    else
    {
        PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
        return(INIT_FAILED);
    }
//--- привязка массивов к индикаторным буферам
    SetIndexBuffer(0,buff,INDICATOR_DATA);
    SetIndexBuffer(1,color_buff,INDICATOR_COLOR_INDEX);
//---- установка значений индикатора, которые не будут видимы на графике
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
    return(INIT_SUCCEEDED);
}

```

```

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    ArraySetAsSeries(time, false);
    //--- цикл для еще необработанных баров
    for(int i=prev_calculated;i<rates_total;i++)
    {
        //--- по умолчанию 0
        buff[i]=0;
        color_buff[i]=0; // красный цвет по умолчанию
        //--- проверка, есть ли еще данные
        if(ind<size)
        {
            for(int j=ind;j<size;j++)
            {
                //--- если даты совпадают, то используем значение из файла
                if(time[i]==time_buff[j])
                {
                    //--- получим цену
                    buff[i]=close_buff[j];
                    //--- если текущая цена больше предыдущей, то цвет синий
                    if(buff[i-1]>buff[i])
                        color_buff[i]=1;
                    //--- увеличим счетчик
                    ind=j+1;
                    break;
                }
            }
        }
        //--- return value of prev_calculated for next call
        return(rates_total);
    }
}

```

#### Смотри также

[Вещественные типы \(double, float\)](#), [FileReadDouble](#), [FileWriteFloat](#)

## FileReadInteger

Читает из бинарного файла значение типа int, short или char в зависимости от указанной длины в байтах. Чтение производится с текущего положения файлового указателя.

```
int FileReadInteger(
    int file_handle,           // handle файла
    int size=INT_VALUE         // размер целого типа в байтах
);
```

### Параметры

*file\_handle*

[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*size=INT\_VALUE*

[in] Количество байт (до 4 включительно), которые нужно прочитать. Предусмотрены соответствующие константы: CHAR\_VALUE=1, SHORT\_VALUE=2 и INT\_VALUE=4, таким образом функция может прочитать целое значение типа char, short или int.

### Возвращаемое значение

Значение типа int. Результат этой функции необходимо явно приводить к целевому типу, то есть тому типу данных, который требуется прочитать. Так как возвращается значение типа int, то его можно спокойно преобразовать в любое целое значение. Файловый указатель перемещается на количество считываемых байт.

### Примечание

При чтении менее 4-х байт, полученный результат всегда будет положительным. Если читается один или два байта, то знак числа можно точно определить путем явного приведения соответственно к типу char (1 байт) или типу short (2 байта). Получение знака для трехбайтового числа является нетривиальным, так как нет соответствующего [базового типа](#).

**Пример** (используется файл, полученный в результате работы примера для функции [FileWriteInteger](#))

```
//+-----+
//|                               Demo_FileReadInteger.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots    1
//---- plot Label1
#property indicator_label1  "Trends"
#property indicator_type1   DRAW_SECTION
#property indicator_color1  clrRed
```

```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- параметры для чтения данных
input string InpFileName="Trend.bin"; // имя файла
input string InpDirectoryName="Data"; // имя директории
//--- глобальные переменные
int      ind=0;
int      size=0;
datetime time_buff[];
//--- indicator buffers
double   buff[];
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
    int def_size=100;
//--- выделим память для массива
    ArrayResize(time_buff,def_size);
//--- откроем файл
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("Файл %s открыт для чтения",InpFileName);
        PrintFormat("Путь к файлу: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
//--- вспомогательные переменные
        int arr_size;
        uchar arr[];
//--- прочитаем данные из файла
        while(!FileIsEnding(file_handle))
        {
            //--- узнаем сколько символов использовано для записи времени
            arr_size=FileReadInteger(file_handle,INT_VALUE);
            ArrayResize(arr,arr_size);
            for(int i=0;i<arr_size;i++)
                arr[i]=(char)FileReadInteger(file_handle,CHAR_VALUE);
//--- запомним значение времени
            time_buff[size]=StringToTime(CharArrayToString(arr));
            size++;
//--- увеличим размеры массивов если они переполнены
            if(size==def_size)
            {
                def_size+=100;
                ArrayResize(time_buff,def_size);
            }
        }
//--- закроем файл
        FileClose(file_handle);
    }
}

```

```

        PrintFormat("Данные прочитаны, файл %s закрыт", InpFileName);
    }
else
{
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d", InpFileName, GetLastError());
    return(INIT_FAILED);
}

//--- привязка массива к индикаторному буферу
SetIndexBuffer(0,buff,INDICATOR_DATA);
//--- установка значений индикатора, которые не будут видимы на графике
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---

return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    ArraySetAsSeries(time,false);
    ArraySetAsSeries(close,false);
//--- цикл для еще необработанных баров
for(int i=prev_calculated;i<rates_total;i++)
{
    //--- по умолчанию 0
    buff[i]=0;
    //--- проверка, есть ли еще данные
    if(ind<size)
    {
        for(int j=ind;j<size;j++)
        {
            //--- если даты совпадают, то используем значение из файла
            if(time[i]==time_buff[j])
            {
                //--- получим цену
                buff[i]=close[i];
                //--- увеличим счетчик
                ind=j+1;
                break;
            }
        }
    }
}

```

```
        }
    }
}

//--- return value of prev_calculated for next call
return(rates_total);
}
```

#### Смотри также

[IntegerToString](#), [StringToInteger](#), [Целые типы](#), [FileWriteInteger](#)

## FileReadLong

Читает целое число типа long (8 байт) из текущей позиции бинарного файла.

```
long FileReadLong(
    int file_handle          // handle файла
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Значение типа long.

**Пример** (используется файл, полученный в результате работы примера для функции [FileWriteLong\(\)](#))

```
//+-----+
//|                               Demo_FileReadLong.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Label1
#property indicator_label1  "Volume"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrYellow
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_separate_window
//--- параметры для чтения данных
input string InpFileName="Volume.bin"; // имя файла
input string InpDirectoryName="Data"; // имя директории
//--- глобальные переменные
int      ind=0;
int      size=0;
long     volume_buff[];
datetime time_buff[];
//--- indicator buffers
double   buff[];
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
```

```

{
//--- откроем файл
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("Файл %s открыт для записи",InpFileName);
    PrintFormat("Путь к файлу: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
//--- сначала прочитаем сколько всего в файле данных
size=(int)FileReadLong(file_handle);
//--- выделим память для массивов
ArrayResize(volume_buff,size);
ArrayResize(time_buff,size);
//--- прочитаем данные из файла
for(int i=0;i<size;i++)
{
    time_buff[i]=(datetime)FileReadLong(file_handle);
    volume_buff[i]=FileReadLong(file_handle);
}
//--- закроем файл
FileClose(file_handle);
PrintFormat("Данные прочитаны, файл %s закрыт",InpFileName);
}
else
{
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
    return(INIT_FAILED);
}
//--- привязка массива к индикаторному буферу с индексом 0
SetIndexBuffer(0,buff,INDICATOR_DATA);
//--- установка значений индикатора, которые не будут видимы на графике
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{

```

```
        ArraySetAsSeries(time, false);
//--- цикл для еще необработанных баров
for(int i=prev_calculated;i<rates_total;i++)
{
    //--- по умолчанию 0
    buff[i]=0;
    //--- проверка, есть ли еще данные
    if(ind<size)
    {
        for(int j=ind;j<size;j++)
        {
            //--- если даты совпадают, то используем значение из файла
            if(time[i]==time_buff[j])
            {
                buff[i]=(double)volume_buff[j];
                ind=j+1;
                break;
            }
        }
    }
//--- return value of prev_calculated for next call
return(rates_total);
}
```

#### Смотри также

[Целые типы](#), [FileReadInteger](#), [FileWriteLong](#)

## FileReadNumber

Читает из файла типа CSV строку от текущего положения до разделителя (либо до конца текстовой строки) и преобразует прочитанную строку в значение типа double.

```
double FileReadNumber(
    int file_handle // handle файла
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Значение типа double.

**Пример** (используется файл, полученный в результате работы примера для функции [FileWriteString](#))

```
//+-----+
//|                               Demo_FileReadNumber.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//---- plot Label1
#property indicator_label1 "Overbought & Oversold"
#property indicator_type1 DRAW_COLOR_BARS
#property indicator_color1 clrRed, clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 2
//--- параметры для чтения данных
input string InpFileName="RSI.csv"; // имя файла
input string InpDirectoryName="Data"; // имя директории
//--- индикаторные буферы
double open_buff[];
double high_buff[];
double low_buff[];
double close_buff[];
double color_buff[];
//--- переменные перекупленности
int ovb_ind=0;
int ovb_size=0;
datetime ovb_time[];
```

```

//--- переменные перепроданности
int      ovs_ind=0;
int      ovs_size=0;
datetime ovs_time[];

//+-----+
//| Custom indicator initialization function           |
//+-----+

int OnInit()
{
//--- переменные размеров массивов по умолчанию
    int ovb_def_size=100;
    int ovs_def_size=100;
//--- выделим память для массивов
    ArrayResize(ovb_time,ovb_def_size);
    ArrayResize(ovs_time,ovs_def_size);
//--- откроем файл
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV|FILE_TEXT);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("Файл %s открыт для чтения",InpFileName);
        PrintFormat("Путь к файлу: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
        double value;
//--- читаем данные из файла
        while(!FileIsEnding(file_handle))
        {
            //--- читаем первое значение в строке
            value=FileReadNumber(file_handle);
            //--- читаем в разные массивы в зависимости от результата функции
            if(value>=70)
                ReadData(file_handle,ovb_time,ovb_size,ovb_def_size);
            else
                ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
        }
//--- закроем файл
        FileClose(file_handle);
        PrintFormat("Данные прочитаны, файл %s закрыт",InpFileName);
    }
    else
    {
        PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
        return(INIT_FAILED);
    }
//--- привязка массивов
    SetIndexBuffer(0,open_buff,INDICATOR_DATA);
    SetIndexBuffer(1,high_buff,INDICATOR_DATA);
    SetIndexBuffer(2,low_buff,INDICATOR_DATA);
    SetIndexBuffer(3,close_buff,INDICATOR_DATA);
    SetIndexBuffer(4,color_buff,INDICATOR_COLOR_INDEX);
}

```

```

//---- установка значений индикатора, которые не будут видимы на графике
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
return(INIT_SUCCEEDED);
}

//+-----+
//| Чтение данных строки файла |
//+-----+
void ReadData(const int file_handle,datetime &arr[],int &size,int &def_size)
{
    bool flag=false;
//--- читаем пока не дойдем до конца строки или файла
while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
{
    //--- сдвинем каретку, прочитав число
    if(flag)
        FileReadNumber(file_handle);
    //--- запоминаем текущую дату
    arr[size]=FileReadDatetime(file_handle);
    size++;
    //--- при необходимости увеличим размер массива
    if(size==def_size)
    {
        def_size+=100;
        ArrayResize(arr,def_size);
    }
    //--- проскочили первую итерацию
    flag=true;
}
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    ArraySetAsSeries(time,false);
    ArraySetAsSeries(open,false);
    ArraySetAsSeries(high,false);
    ArraySetAsSeries(low,false);
    ArraySetAsSeries(close,false);
}

```

```

//--- цикл для еще необработанных баров
for(int i=prev_calculated;i<rates_total;i++)
{
    //--- по умолчанию 0
    open_buff[i]=0;
    high_buff[i]=0;
    low_buff[i]=0;
    close_buff[i]=0;
    color_buff[i]=0;
    //--- проверка, есть ли еще данные
    if(ovb_ind<ovb_size)
        for(int j=ovb_ind;j<ovb_size;j++)
    {
        //--- если даты совпадают, то бар лежит в зоне перекупленности
        if(time[i]==ovb_time[j])
        {
            open_buff[i]=open[i];
            high_buff[i]=high[i];
            low_buff[i]=low[i];
            close_buff[i]=close[i];
            //--- 0 - красный цвет
            color_buff[i]=0;
            //--- увеличим счетчик
            ovb_ind=j+1;
            break;
        }
    }
    //--- проверка, есть ли еще данные
    if(ovs_ind<ovs_size)
        for(int j=ovs_ind;j<ovs_size;j++)
    {
        //--- если даты совпадают, то бар лежит в зоне перепроданности
        if(time[i]==ovs_time[j])
        {
            open_buff[i]=open[i];
            high_buff[i]=high[i];
            low_buff[i]=low[i];
            close_buff[i]=close[i];
            //--- 1 - синий цвет
            color_buff[i]=1;
            //--- увеличим счетчик
            ovs_ind=j+1;
            break;
        }
    }
}
//--- return value of prev_calculated for next call
return(rates_total);
}

```

```
//+-----+
//| Обработчик события ChartEvent |
//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam
)
{
//--- Изменяем толщину индикатора в зависимости от масштаба
if(ChartGetInteger(0,CHART_SCALE)>3)
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,2);
else
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
}
```

Смотри также

[FileWriteString](#)

## FileReadString

Читает из файла строку с текущего положения файлового указателя.

```
string FileReadString(
    int file_handle,          // handle файла
    int length=-1            // длина строки
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).  
*length=-1*  
 [in] Количество символов для чтения.

### Возвращаемое значение

Прочитанная строка(string).

### Примечание

При чтении из bin-файла обязательно указывать длину читаемой строки. При чтении из txt-файла длину строки указывать не надо, будет прочитана строка от текущего положения до признака перевода строки "\r\n". При чтении из csv-файла длину строки также указывать не надо, будет прочитана строка от текущего положения до ближайшего разделителя либо до признака конца текстовой строки.

Если файл открыт с [флагом FILE\\_ANSI](#), то прочитанная строка преобразовывается в Unicode.

**Пример** (используется файл, полученный в результате работы примера для функции [FileWriteInteger\(\)](#))

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- параметры для чтения данных
input string InpFileName="Trend.bin"; // имя файла
input string InpDirectoryName="Data"; // имя директории
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- откроем файл
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN|FILE_TEXT);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("Файл %s открыт для чтения",InpFileName);
        PrintFormat("Путь к файлу: %s\\Files\\\",TerminalInfoString(TERMINAL_DATA_PATH));
//--- вспомогательные переменные
        int str_size;
```

```
string str;
//--- прочитаем данные из файла
while(!FileIsEnding(file_handle))
{
    //--- узнаем сколько символов использовано для записи времени
    str_size=FileReadInteger(file_handle, INT_VALUE);
    //--- прочитаем строку
    str=FileReadString(file_handle,str_size);
    //--- распечатаем строку
    PrintFormat(str);
}
//--- закроем файл
FileClose(file_handle);
PrintFormat("Данные прочитаны, файл %s закрыт", InpFileName);
}
else
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d", InpFileName, GetLastError());
}
```

#### Смотри также

[Тип string](#), [Преобразование данных](#), [FileWriteInteger](#)

## FileReadStruct

Считывает из бинарного файла содержимое в структуру, переданную в качестве параметра. Чтение производится с текущего положения файлового указателя.

```
uint FileReadStruct(
    int      file_handle,           // handle файла
    const void& struct_object,     // структура, куда происходит считывание
    int      size=-1               // размер структуры в байтах
);
```

### Параметры

*file\_handle*

[in] Файловый описатель открытого бинарного файла.

*struct\_object*

[out] Ссылка на объект указанной структуры. Структура не должна содержать строки, [динамические массивы](#), [виртуальные функции](#), а также указатели на объекты и функции.

*size=-1*

[in] Количество байт, которые нужно прочитать. Если размер не указан или указано большее количество байт, чем размер структуры, то используется точный размер указанной структуры.

### Возвращаемое значение

В случае удачи функция возвращает количество прочитанных байт. Файловый указатель перемещается на это же количество байт.

**Пример** (используется файл, полученный в результате работы примера для функции [FileWriteStruct](#))

```
//+-----+
//|                               Demo_FileReadStruct.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   1
//--- plot Label1
#property indicator_label1  "Candles"
#property indicator_type1   DRAW_CANDLES
#property indicator_color1  clrOrange
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_separate_window
//--- параметры для получения данных
input string  InpFileName="EURUSD.txt"; // имя файла
```

```

input string  InpDirectoryName="Data"; // имя директории
//+-----+
//| Структура для хранения данных свечи |
//+-----+
struct candlesticks
{
    double          open;   // цена открытия
    double          close;  // цена закрытия
    double          high;   // максимальная цена
    double          low;    // минимальная цена
    datetime        date;   // дата
};

//--- индикаторные буферы
double      open_buff[];
double      close_buff[];
double      high_buff[];
double      low_buff[];

//--- глобальные переменные
candlesticks cand_buff[];
int         size=0;
int         ind=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    int default_size=100;
    ArrayResize(cand_buff,default_size);
//--- откроем файл
    ResetLastError();
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN|FILE_TEXT);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("Файл %s открыт для чтения",InpFileName);
        PrintFormat("Путь к файлу: %s\\Files\\\",TerminalInfoString(TERMINAL_COMMONDATA_INFORMATION));
//--- прочитаем данные из файла
        while(!FileIsEnding(file_handle))
        {
            //--- запишем данные в массив
            FileReadStruct(file_handle,cand_buff[size]);
            size++;
            //--- проверим массив на переполненность
            if(size==default_size)
            {
                //--- увеличим размерность массива
                default_size+=100;
                ArrayResize(cand_buff,default_size);
            }
        }
    }
}

```

```

//--- закроем файл
FileClose(file_handle);
PrintFormat("Данные прочитаны, файл %s закрыт", InpFileName);
}
else
{
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d", InpFileName, GetLastError());
    return(INIT_FAILED);
}

//--- indicator buffers mapping
SetIndexBuffer(0,open_buff,INDICATOR_DATA);
SetIndexBuffer(1,high_buff,INDICATOR_DATA);
SetIndexBuffer(2,low_buff,INDICATOR_DATA);
SetIndexBuffer(3,close_buff,INDICATOR_DATA);

//--- пустое значение
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---

return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    ArraySetAsSeries(time, false);
//--- цикл для еще необработанных свечек
    for(int i=prev_calculated;i<rates_total;i++)
    {
        //--- по умолчанию 0
        open_buff[i]=0;
        close_buff[i]=0;
        high_buff[i]=0;
        low_buff[i]=0;
//--- проверка, есть ли еще данные
        if(ind<size)
        {
            for(int j=ind;j<size;j++)
            {
//--- если даты совпадают, то используем значение из файла
                if(time[i]==cand_buff[j].date)

```

```
{  
    open_buff[i]=cand_buff[j].open;  
    close_buff[i]=cand_buff[j].close;  
    high_buff[i]=cand_buff[j].high;  
    low_buff[i]=cand_buff[j].low;  
    //--- увеличиваем счетчик  
    ind=j+1;  
    break;  
}  
}  
}  
//--- return value of prev_calculated for next call  
return(rates_total);  
}
```

Смотри также

[Структуры и классы](#), [FileWriteStruct](#)

## FileSeek

Перемещает положение файлового указателя на указанное количество байт относительно указанного положения.

```
bool FileSeek(
    int          file_handle,      // handle файла
    long         offset,           // в байтах
    ENUM_FILE_POSITION origin       // позиция для отсчета
);
```

### Параметры

*file\_handle*

[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*offset*

[in] Смещение в байтах (может принимать и отрицательное значение).

*origin*

[in] Точка отсчета для смещения. Может принимать одно из значений перечисления [ENUM\\_FILE\\_POSITION](#).

### Возвращаемое значение

В случае удачи функция возвращает `true`, в случае ошибки - `false`. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Если результатом выполнения функции `FileSeek()` является отрицательное смещение (выход за "левую границу" файла), то файловый указатель будет установлен на начало файла.

Если выставить позицию за "правую границу" файла (больше, чем размер файла), то последующая запись в файл будет произведена не с конца файла, а с выставленной позиции. При этом между предыдущим концом файла и выставленной позицией будут записаны неопределенные значения.

### Пример:

```
//+-----+
//|                               Demo_FileSeek.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input string InpFileName="file.txt";      // имя файла
input string InpDirectoryName="Data";      // имя директории
```

```

input int      InpEncodingType=FILE_ANSI; // ANSI=32 или UNICODE=64
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- установим значение переменной для генерации случайных чисел
    _RandomSeed=GetTickCount();
//--- переменные для позиций начала строк
    ulong pos[];
    int    size;
//--- сбросим значение ошибки
    ResetLastError();
//--- откроем файл
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpEr
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("Файл %s открыт для чтения",InpFileName);
        //--- получим позицию начала для каждой строки в файле
        GetStringPositions(file_handle,pos);
        //--- определим сколько всего строк в файле
        size=ArraySize(pos);
        if(!size)
        {
            //--- если в файле нет строк, то завершаем работу
            PrintFormat("Файл %s пуст!",InpFileName);
            FileClose(file_handle);
            return;
        }
        //--- выберем случайно номер строки
        int ind=MathRand()%size;
        //--- сдвинем позицию на начало этой строки
        if(FileSeek(file_handle,pos[ind],SEEK_SET)==true)
        {
            //--- прочитаем и распечатаем строку с номером ind
            PrintFormat("Текст строки с номером %d: \"%s\"",ind,FileReadString(file_handl
            }
        //--- закроем файл
        FileClose(file_handle);
        PrintFormat("Файл %s закрыт",InpFileName);
    }
    else
        PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
}
//+-----+
//| Функция определяет позиции начала для каждой из строк в файле и   |
//| помещает их в массив arr                                         |
//+-----+
void GetStringPositions(const int handle,ulong &arr[])

```

```
{  
//--- размер массива по умолчанию  
int def_size=127;  
//--- выделим память для массива  
ArrayResize(arr,def_size);  
//--- счетчик строк  
int i=0;  
//--- если не конец файла, то есть хотя бы одна строка  
if(!FileIsEnding(handle))  
{  
    arr[i]=FileTell(handle);  
    i++;  
}  
else  
    return; // файл пуст, выходим  
//--- определим сдвиг в байтах в зависимости от кодировки  
int shift;  
if(FileGetInteger(handle,FILE_IS_ANSI))  
    shift=1;  
else  
    shift=2;  
//--- в цикле перебираем строки  
while(1)  
{  
    //--- читаем строку  
    FileReadString(handle);  
    //--- проверка на конец файла  
    if(!FileIsEnding(handle))  
    {  
        //--- запомним позицию следующей строки  
        arr[i]=FileTell(handle)+shift;  
        i++;  
        //--- увеличим размер массива, если он переполнен  
        if(i==def_size)  
        {  
            def_size+=def_size+1;  
            ArrayResize(arr,def_size);  
        }  
    }  
    else  
        break; // конец файла, выходим  
}  
//--- установим истинный размер массива  
ArrayResize(arr,i);  
}
```

## FileSize

Возвращает размер файла в байтах.

```
ulong  FileSize(
    int   file_handle        // handle файла
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Значение типа `int`.

### Примечание

Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Пример:

```
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input ulong  InpThresholdSize=20;           // граница размера файлов в килобайтах
input string InpBigFolderName="big";         // папка для больших файлов
input string InpSmallFolderName="small";      // папка для маленьких файлов
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
    string   file_name;          // переменная для хранения имен файлов
    string   filter=".csv";       // фильтр для поиска файлов
    ulong    file_size=0;         // размер файла в байтах
    int     size=0;              // количество файлов
//--- распечатаем путь к папке в которой будем работать
    PrintFormat("Работаем в папке %s\\Files\\",TerminalInfoString(TERMINAL_COMMONDATA_I
//--- получение хэндла поиска в корне общей папки всех терминалов
    long search_handle=FileFindFirst(filter,file_name,FILE_COMMON);
//--- проверим, успешно ли отработала функция FileFindFirst()
    if(search_handle!=INVALID_HANDLE)
    {
//--- в цикле перемещаем файлы в зависимости от их размера
        do
        {
//--- откроем файл
            ResetLastError();
            int file_handle=FileOpen(file_name,FILE_READ|FILE_CSV|FILE_COMMON);
            if(file_handle!=INVALID_HANDLE)
```

```
{  
    //--- получим размер файла  
    file_size=FileSize(file_handle);  
    //--- закроем файл  
    FileClose(file_handle);  
}  
  
else  
{  
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",file_name,GetLastError());  
    continue;  
}  
  
//--- распечатаем размер файла  
PrintFormat("Размер файла %s равен %d байт",file_name,file_size);  
//--- определим путь для перемещения файла  
string path;  
if(file_size>InpThresholdSize*1024)  
    path=InpBigFolderName+"//"+file_name;  
else  
    path=InpSmallFolderName+"//"+file_name;  
//--- переместим файл  
ResetLastError();  
if(FileMove(file_name,FILE_COMMON,path,FILE_REWRITE|FILE_COMMON))  
    PrintFormat("Файл %s перемещен",file_name);  
else  
    PrintFormat("Ошибка, код = %d",GetLastError());  
}  
  
while(FileFindNext(search_handle,file_name));  
//--- закрываем хэндл поиска  
FileFindClose(search_handle);  
}  
else  
    Print("Files not found!");  
}
```

## FileTell

Возвращает текущее положение файлового указателя соответствующего открытого файла.

```
ulong FileTell(
    int file_handle // handle файла
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

### Возвращаемое значение

Текущая позиция файлового описателя в байтах от начала файла.

### Примечание

Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Пример:

```
//+-----+
//|                               Demo_FileTell.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input string InpFileName="file.txt";      // имя файла
input string InpDirectoryName="Data";       // имя директории
input int    InpEncodingType=FILE_ANSI; // ANSI=32 или UNICODE=64
//+-----+
//| Script program start function          |
//+-----+
void OnStart()
{
//--- установим значение переменной для генерации случайных чисел
    _RandomSeed=GetTickCount();
//--- переменные для позиций начала строк
    ulong pos[];
    int    size;
//--- сбросим значение ошибки
    ResetLastError();
//--- откроем файл
    int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpEr
    if(file_handle!=INVALID_HANDLE)
```

```

{
    PrintFormat("Файл %s открыт для чтения", InpFileName);
    //--- получим позицию начала для каждой строки в файле
    GetStringPositions(file_handle, pos);
    //--- определим сколько всего строк в файле
    size=ArraySize(pos);
    if(!size)
    {
        //--- если в файле нет строк, то завершаем работу
        PrintFormat("Файл %s пуст!", InpFileName);
        FileClose(file_handle);
        return;
    }
    //--- выберем случайно номер строки
    int ind=MathRand()%size;
    //--- сдвинем позицию на начало этой строки
    FileSeek(file_handle, pos[ind], SEEK_SET);
    //--- прочитаем и распечатаем строку с номером ind
    PrintFormat("Текст строки с номером %d: \"%s\"", ind, FileReadString(file_handle));
    //--- закроем файл
    FileClose(file_handle);
    PrintFormat("Файл %s закрыт", InpFileName);
}
else
{
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d", InpFileName, GetLastError());
}

//+-----+
//| Функция определяет позиции начала для каждой из строк в файле и |
//| помещает их в массив arr                                         |
//+-----+
void GetStringPositions(const int handle, ulong &arr[])
{
    //--- размер массива по умолчанию
    int def_size=127;
    //--- выделим память для массива
    ArrayResize(arr, def_size);
    //--- счетчик строк
    int i=0;
    //--- если не конец файла, то есть хотя бы одна строка
    if(!FileIsEnding(handle))
    {
        arr[i]=FileTell(handle);
        i++;
    }
    else
        return; // файл пуст, выходим
    //--- определим сдвиг в байтах в зависимости от кодировки
    int shift;
    if(FileGetInteger(handle, FILE_IS_ANSI))

```

```
    shift=1;
else
    shift=2;
//--- в цикле перебираем строки
while(1)
{
    //--- читаем строку
    FileReadString(handle);
    //--- проверка на конец файла
    if(!FileIsEnding(handle))
    {
        //--- запомним позицию следующей строки
        arr[i]=FileTell(handle)+shift;
        i++;
        //--- увеличим размер массива, если он переполнен
        if(i==def_size)
        {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
        }
    }
    else
        break; // конец файла, выходим
}
//--- установим истинный размер массива
ArrayResize(arr,i);
}
```

## FileWrite

Запись данных в файл типа CSV или TXT, разделитель между данными вставляется автоматически, если он не равен 0. После записи в файл добавляется признак конца строки "\r\n".

```
uint FileWrite(
    int file_handle, // handle файла
    ...
    // список записываемых параметров
);
```

### Параметры

*file\_handle*

[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

...

[in] Список параметров, разделенных запятыми, для записи в файл. Количество выводимых в файл параметров не должно превышать 63.

### Возвращаемое значение

Количество записанных байт.

### Примечание

При выводе числовые данные преобразуются в текстовый формат (см. функцию [Print\(\)](#)). Данные типа double выводятся с точностью до 16 десятичных цифр после точки, при этом данные могут выводиться либо в традиционном либо в научном формате - в зависимости от того, как запись будет наиболее компактна. Данные типа float выводятся с 5 десятичными цифрами после точки. Для вывода вещественных чисел с другой точностью либо в явно указанном формате необходимо использовать функцию [DoubleToString\(\)](#).

Числа типа bool выводятся в виде строк "true" или "false". Числа типа datetime выводятся в формате "YYYY.MM.DD HH:MI:SS".

### Пример:

```
//+-----+
//|                               Demo_FileWrite.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- параметры для получения данных из терминала
input string          InpSymbolName="EURUSD";           // валюта пара
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // таймфрейм
input int              InpFastEMAPeriod=12;             // период быстрой средней
input int              InpSlowEMAPeriod=26;             // период медленной средней
input int              InpSignalPeriod=9;                // период усреднения разнос
```

```

input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;           // тип цены
input datetime          InpDateStart=D'2012.01.01 00:00'; // дата начала копирования
//--- параметры для записи данных в файл
input string            InpFileName="MACD.csv";   // имя файла
input string            InpDirectoryName="Data"; // имя директории
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    datetime date_finish; // дата конца копирования данных
    bool sign_buff[]; // массив сигналов (true - покупка, false - продажа)
    datetime time_buff[]; // массив времени наступления сигналов
    int sign_size=0; // размер массивов сигналов
    double macd_buff[]; // массив значений индикатора
    datetime date_buff[]; // массив дат индикатора
    int macd_size=0; // размер массивов индикатора
//--- время окончания - текущее
    date_finish=TimeCurrent();
//--- получим хэндл индикатора MACD
    ResetLastError();
    int macd_handle=iMACD(InpSymbolName,InpSymbolPeriod,InpFastEMAPeriod,InpSlowEMAPeriod);
    if(macd_handle==INVALID_HANDLE)
    {
        //--- не удалось получить хэндл индикатора
        PrintFormat("Ошибка получения хэндла индикатора. Код ошибки = %d",GetLastError());
        return;
    }
//--- находимся в цикле, пока индикатор не рассчитает все свои значения
    while(BarsCalculated(macd_handle)==-1)
        Sleep(10); // задержка, чтобы индикатор успел вычислить свои значения
//--- скопируем значения индикатора за определенный период
    ResetLastError();
    if(CopyBuffer(macd_handle,0,InpDateStart,date_finish,macd_buff)==-1)
    {
        PrintFormat("Не удалось скопировать значения индикатора. Код ошибки = %d",GetLastError());
        return;
    }
//--- скопируем соответствующее время для значений индикатора
    ResetLastError();
    if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,date_buff)==-1)
    {
        PrintFormat("Не удалось скопировать значения времени. Код ошибки = %d",GetLastError());
        return;
    }
//--- освободим память, занимаемую индикатором
    IndicatorRelease(macd_handle);
//--- получим размер буфера
    macd_size=ArraySize(macd_buff);
}

```

```

//--- проанализируем данные и сохраним сигналы индикатора в массивы
ArrayResize(sign_buff,macd_size-1);
ArrayResize(time_buff,macd_size-1);
for(int i=1;i<macd_size;i++)
{
    //--- сигнал на покупку
    if(macd_buff[i-1]<0 && macd_buff[i]>=0)
    {
        sign_buff[sign_size]=true;
        time_buff[sign_size]=date_buff[i];
        sign_size++;
    }
    //--- сигнал на продажу
    if(macd_buff[i-1]>0 && macd_buff[i]<=0)
    {
        sign_buff[sign_size]=false;
        time_buff[sign_size]=date_buff[i];
        sign_size++;
    }
}
//--- откроем файл для записи значений индикатора (если его нет, то создастся автоматически)
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("Файл %s открыт для записи",InpFileName);
    PrintFormat("Путь к файлу: %s\\Files\\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- сначала запишем количество сигналов
    FileWrite(file_handle,sign_size);
    //--- запишем время сигналов и их значения в файл
    for(int i=0;i<sign_size;i++)
        FileWrite(file_handle,time_buff[i],sign_buff[i]);
    //--- закрываем файл
    FileClose(file_handle);
    PrintFormat("Данные записаны, файл %s закрыт",InpFileName);
}
else
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
}

```

#### Смотри также

[Comment](#), [Print](#), [StringFormat](#)

## FileWriteArray

Записывает в бинарный файл массивы любых типов, кроме строковых (может быть массив структур, не содержащих строки и динамические массивы).

```
uint FileWriteArray(
    int      file_handle,           // handle файла
    const void& array[],           // массив
    int      start=0,              // начальный индекс в массиве
    int      count=WHOLE_ARRAY     // количество элементов
);
```

### Параметры

*file\_handle*

[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*array[]*

[out] Массив для записи.

*start=0*

[in] Начальный индекс в массиве (номер первого записываемого элемента).

*count=WHOLE\_ARRAY*

[in] Количество записываемых элементов ([WHOLE\\_ARRAY](#) означает, что записываются все элементы начиная с номера start до конца массива).

### Возвращаемое значение

Количество записанных элементов.

### Примечание

Строковый массив может записываться только в файл типа TXT. В этом случае строки автоматически завершаются символами конца строки "\r\n". В зависимости от типа файла ANSI или UNICODE, строки преобразовываются к ansi-кодировке, или нет.

### Пример:

```
//-----+
//|                               Demo_FileWriteArray.mq5 |
//| Copyright 2013, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- входные параметры
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//-----+
//| Структура для хранения данных о ценах |
//-----+
```

```

struct prices
{
    datetime date; // дата
    double bid; // цена бид
    double ask; // цена аск
};

//--- глобальные переменные
int count=0;
int size=20;
string path=InpDirectoryName+"//"+InpFileName;
prices arr[];

//+-----+
//| Expert initialization function |
//+-----+
int OnInit()
{
//--- выделение памяти для массива
    ArrayResize(arr,size);
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//--- запись оставшихся count строк, если count<n
    WriteData(count);
}
//+-----+
//| Expert tick function |
//+-----+
void OnTick()
{
//--- сохраним данные в массив
    arr[count].date=TimeCurrent();
    arr[count].bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    arr[count].ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- отобразим текущие данные
    Print("Date = ",arr[count].date," Bid = ",arr[count].bid," Ask = ",arr[count].ask);
//--- увеличим счетчик
    count++;
//--- если массив заполнился, то записываем данные в файл и обнуляем его
    if(count==size)
    {
        WriteData(size);
        count=0;
    }
}

```

```
//+-----+
//| Запись n элементов массива в файл
//+-----+
void WriteData(const int n)
{
//--- откроем файл
ResetLastError();
int handle=FileOpen(path,FILE_READ|FILE_WRITE|FILE_BIN);
if(handle!=INVALID_HANDLE)
{
//--- запишем данные массива в конец файла
FileSeek(handle,0,SEEK_END);
FileWriteArray(handle,arr,0,n);
//--- закрываем файл
FileClose(handle);
}
else
Print("Failed to open the file, error ",GetLastError());
}
```

#### Смотри также

[Переменные](#), [FileSeek](#)

## FileWriteDouble

Записывает в бинарный файл значение параметра типа double с текущего положения файлового указателя.

```
uint FileWriteDouble(
    int     file_handle,      // handle файла
    double  value            // значения для записи
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*value*  
 [in] Значение типа double.

### Возвращаемое значение

В случае удачи функция возвращает количество записанных байт (в данном случае `sizeof(double) = 8`). Файловый указатель перемещается на это же количество байт.

### Пример:

```
//+-----+
//|                               Demo_FileWriteDouble.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- параметры для получения данных из терминала
input string          InpSymbolName="EURJPY";           // валюта пара
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_M15;        // таймфрейм
input int              InpMAPeriod=10;                   // период сглаживания
input int              InpMAShift=0;                    // смещение индикатора
input ENUM_MA_METHOD   InpMAMethod=MODE_SMA;             // тип сглаживания
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;    // тип цены
input datetime         InpDateStart=D'2013.01.01 00:00'; // дата начала копирования
//--- параметры для записи данных в файл
input string          InpFileName="MA.csv";           // имя файла
input string          InpDirectoryName="Data";           // имя директории
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
```

```

datetime date_finish=TimeCurrent();
double ma_buff[];
datetime time_buff[];
int size;
//--- получим хэндл индикатора МА
ResetLastError();
int ma_handle=iMA(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpMAShift,InpMAMethod,
if(ma_handle==INVALID_HANDLE)
{
    //--- не удалось получить хэндл индикатора
    PrintFormat("Ошибка получения хэндла индикатора. Код ошибки = %d",GetLastError());
    return;
}
//--- находимся в цикле, пока индикатор не рассчитает все свои значения
while(BarsCalculated(ma_handle)==-1)
    Sleep(20); // задержка, чтобы индикатор успел вычислить свои значения
PrintFormat("В файл будут записаны значения индикатора, начиная с %s",TimeToString
//--- скопируем значения индикатора
ResetLastError();
if(CopyBuffer(ma_handle,0,InpDateStart,date_finish,ma_buff)==-1)
{
    PrintFormat("Не удалось скопировать значения индикатора. Код ошибки = %d",GetLastError());
    return;
}
//--- скопируем время появления соответствующих баров
ResetLastError();
if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
{
    PrintFormat("Не удалось скопировать значения времени. Код ошибки = %d",GetLastError());
    return;
}
//--- получим размер буфера
size=ArraySize(ma_buff);
//--- освободим память, занимаемую индикатором
IndicatorRelease(ma_handle);
//--- откроем файл для записи значений индикатора (если его нет, то создастся автоматически)
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("Файл %s открыт для записи",InpFileName);
    PrintFormat("Путь к файлу: %s\\Files\\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- сначала запишем размер выборки данных
    FileWriteDouble(file_handle,(double)size);
    //--- запишем время и значения индикатора в файл
    for(int i=0;i<size;i++)
    {
        FileWriteDouble(file_handle,(double)time_buff[i]);
        FileWriteDouble(file_handle,ma_buff[i]);
    }
}

```

```
    }
    //--- закрываем файл
    FileClose(file_handle);
    PrintFormat("Данные записаны, файл %s закрыт", InpFileName);
}
else
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d", InpFileName, GetLastError());
}
```

Смотри также

[Вещественные типы \(double, float\)](#)

## FileWriteFloat

Записывает в бинарный файл значение параметра типа float с текущего положения файлового указателя.

```
uint FileWriteFloat(
    int   file_handle,      // handle файла
    float value            // записываемое значение
);
```

### Параметры

*file\_handle*  
 [in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*value*  
 [in] Значение типа float.

### Возвращаемое значение

В случае удачи функция возвращает количество записанных байт (в данном случае `sizeof(float) =4`). Файловый указатель перемещается на это же количество байт.

### Пример:

```
//+-----+
//|                               Demo_FileWriteFloat.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- параметры для получения данных из терминала
input string          InpSymbolName="EURUSD";           // валютная пара
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_M15;       // таймфрейм
input datetime        InpDateStart=D'2013.01.01 00:00'; // дата начала копирования данных
//--- параметры для записи данных в файл
input string          InpFileName="Close.bin"; // имя файла
input string          InpDirectoryName="Data"; // имя директории
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    datetime date_finish=TimeCurrent();
    double  close_buff[];
    datetime time_buff[];
    int     size;
```

```

//--- сбросим значение ошибки
ResetLastError();

//--- скопируем цену закрытия для каждого бара
if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-1)
{
    PrintFormat("Не удалось скопировать значения цен закрытия. Код ошибки = %d",GetLastError());
    return;
}

//--- скопируем время для каждого бара
if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
{
    PrintFormat("Не удалось скопировать значения времени. Код ошибки = %d",GetLastError());
    return;
}

//--- получим размер буфера
size=ArraySize(close_buff);

//--- откроем файл для записи значений (если его нет, то создастся автоматически)
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"\\"+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("Файл %s открыт для записи",InpFileName);
    PrintFormat("Путь к файлу: %s\\Files\\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- запишем время и значения цен закрытия в файл
    for(int i=0;i<size;i++)
    {
        FileWriteDouble(file_handle,(double)time_buff[i]);
        FileWriteFloat(file_handle,(float)close_buff[i]);
    }
    //--- закрываем файл
    FileClose(file_handle);
    PrintFormat("Данные записаны, файл %s закрыт",InpFileName);
}
else
{
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
}

```

#### Смотри также

[Вещественные типы \(double, float\)](#), [FileWriteDouble](#)

## FileWriteInteger

Записывает в бинарный файл значение параметра типа int с текущего положения файлового указателя.

```
uint FileWriteInteger(
    int file_handle,           // handle файла
    int value,                 // записываемое значение
    int size=INT_VALUE         // размер в байтах
);
```

### Параметры

*file\_handle*

[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*value*

[in] Целое значение.

*size=INT\_VALUE*

[in] Количество байт (до 4 включительно), которые нужно записать. Предусмотрены соответствующие константы: CHAR\_VALUE=1, SHORT\_VALUE=2 и INT\_VALUE=4, таким образом функция может записать целое значение типа char, uchar, short, ushort, int или uint

### Возвращаемое значение

В случае удачи функция возвращает количество записанных байт. Файловый указатель перемещается на это же количество байт.

### Пример:

```
//+-----+
//|                               Demo_FileWriteInteger.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- параметры для получения данных из терминала
input string          InpSymbolName="EURUSD";           // валютная пара
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // таймфрейм
input datetime         InpDateStart=D'2013.01.01 00:00'; // дата начала копирования
//--- параметры для записи данных в файл
input string          InpFileName="Trend.bin"; // имя файла
input string          InpDirectoryName="Data"; // имя директории
//+-----+
//| Script program start function
//+-----+
void OnStart()
```

```

{
    datetime date_finish=TimeCurrent();
    double close_buff[];
    datetime time_buff[];
    int size;
//--- сбросим значение ошибки
    ResetLastError();
//--- скопируем цену закрытия для каждого бара
    if(CopyClose(IInpSymbolName,IInpSymbolPeriod,IInpDateStart,date_finish,close_buff)==-1)
    {
        PrintFormat("Не удалось скопировать значения цен закрытия. Код ошибки = %d",GetLastError());
        return;
    }
//--- скопируем время для каждого бара
    if(CopyTime(IInpSymbolName,IInpSymbolPeriod,IInpDateStart,date_finish,time_buff)==-1)
    {
        PrintFormat("Не удалось скопировать значения времени. Код ошибки = %d",GetLastError());
        return;
    }
//--- получим размер буфера
    size=ArraySize(close_buff);
//--- откроем файл для записи значений (если его нет, то создастся автоматически)
    ResetLastError();
    int file_handle=FileOpen(IInpDirectoryName+"\\"+IInpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
    if(file_handle!=INVALID_HANDLE)
    {
        PrintFormat("Файл %s открыт для записи",IInpFileName);
        PrintFormat("Путь к файлу: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PATH));
//---
        int up_down=0; // флаг тенденции
        int arr_size; // размера массива arr
        uchar arr[]; // массив типа uchar
//--- запишем значения времени в файл
        for(int i=0;i<size-1;i++)
        {
            //--- сравним цены закрытия на текущем и следующем барах
            if(close_buff[i]<=close_buff[i+1])
            {
                if(up_down!=1)
                {
                    //--- запишем значение даты в файл, используя FileWriteInteger
                    StringToCharArray(TimeToString(time_buff[i]),arr);
                    arr_size=ArraySize(arr);
                    //--- сначала запишем количество символов в массиве
                    FileWriteInteger(file_handle,arr_size,INT_VALUE);
                    //--- запишем сами символы
                    for(int j=0;j<arr_size;j++)
                        FileWriteInteger(file_handle,arr[j],CHAR_VALUE);
                    //--- изменим флаг тенденции
                }
            }
        }
    }
}

```

```
        up_down=1;
    }
}
else
{
    if(up_down!=-1)
    {
        //--- запишем значение даты в файл, используя FileWriteInteger
        StringToCharArray(TimeToString(time_buff[i]),arr);
        arr_size=ArraySize(arr);
        //--- сначала запишем количество символов в массиве
        FileWriteInteger(file_handle,arr_size,INT_VALUE);
        //--- запишем сами символы
        for(int j=0;j<arr_size;j++)
            FileWriteInteger(file_handle,arr[j],CHAR_VALUE);
        //--- изменим флаг тенденции
        up_down=-1;
    }
}
//--- закрываем файл
FileClose(file_handle);
PrintFormat("Данные записаны, файл %s закрыт",InpFileName);
}
else
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
}
```

#### Смотри также

[IntegerToString](#), [StringToInteger](#), [Целые типы](#)

## FileWriteLong

Записывает в бинарный файл значение параметра типа long с текущего положения файлового указателя.

```
uint FileWriteLong(
    int   file_handle,      // handle файла
    long  value            // записываемое значение
);
```

### Параметры

*file\_handle*

[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*value*

[in] Значение типа long.

### Возвращаемое значение

В случае удачи функция возвращает количество записанных байт (в данном случае `sizeof(long) = 8`). Файловый указатель перемещается на это же количество байт.

### Пример:

```
//+-----+
//|                               Demo_FileWriteLong.mq5  |
//|                               Copyright 2013, MetaQuotes Software Corp.  |
//|                               https://www.mql5.com  |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- параметры для получения данных из терминала
input string          InpSymbolName="EURUSD";           // валютная пара
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // таймфрейм
input datetime         InpDateStart=D'2013.01.01 00:00'; // дата начала копирования данных
//--- параметры для записи данных в файл
input string          InpFileName="Volume.bin"; // имя файла
input string          InpDirectoryName="Data"; // имя директории
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    datetime date_finish=TimeCurrent();
    long    volume_buff[];
    datetime time_buff[];
    int     size;
```

```

//--- сбросим значение ошибки
ResetLastError();

//--- скопируем тиковые объемы для каждого бара
if(CopyTickVolume(IInpSymbolName,IInpSymbolPeriod,IInpDateStart,date_finish,volume_buff))
{
    PrintFormat("Не удалось скопировать значения тикового объема. Код ошибки = %d",GetLastError());
    return;
}

//--- скопируем время для каждого бара
if(CopyTime(IInpSymbolName,IInpSymbolPeriod,IInpDateStart,date_finish,time_buff)==-1)
{
    PrintFormat("Не удалось скопировать значения времени. Код ошибки = %d",GetLastError());
    return;
}

//--- получим размер буфера
size=ArraySize(volume_buff);

//--- откроем файл для записи значений индикатора (если его нет, то создастся автоматически)
ResetLastError();
int file_handle=FileOpen(IInpDirectoryName+"\\"+IInpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("Файл %s открыт для записи",IInpFileName);
    PrintFormat("Путь к файлу: %s\\Files\\\",TerminalInfoString(TERMINAL_DATA_PATH));
    //--- сначала запишем размер выборки данных
    FileWriteLong(file_handle,(long)size);
    //--- запишем время и значения объема в файл
    for(int i=0;i<size;i++)
    {
        FileWriteLong(file_handle,(long)time_buff[i]);
        FileWriteLong(file_handle,volume_buff[i]);
    }
    //--- закрываем файл
    FileClose(file_handle);
    PrintFormat("Данные записаны, файл %s закрыт",IInpFileName);
}
else
{
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",IInpFileName,GetLastError());
}

```

#### Смотри также

[Целые типы](#), [FileWriteInteger](#)

## FileWriteString

Записывает в файл типа BIN, CSV или TXT значение параметра типа string с текущего положения файлового указателя. При записи в файл типа CSV или TXT, если в строке присутствует символ '\n' (LF) без предшествующего символа '\r' (CR), то перед символом '\n' дописывается отсутствующий символ '\r'.

```
uint FileWriteString(
    int         file_handle,      // handle файла
    const string text_string,    // записываемая строка
    int         length=-1        // количество символов
);
```

### Параметры

*file\_handle*

[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*text\_string*

[in] Стока.

*length=-1*

[in] Количество символов, которые нужно записать. Параметр необходим для записи строки в файл типа BIN. Если размер не указан, то записывается вся строка без завершающего 0. Если указан размер меньший, чем длина строки, то записывается часть строки без завершающего 0. Если указан размер больший, чем длина строки, то строка дописывается соответствующим количеством нулей. Для файлов типа CSV и TXT этот параметр игнорируется и строка записывается полностью.

### Возвращаемое значение

В случае удачи функция возвращает количество записанных байт. Файловый указатель перемещается на это же количество байт.

### Примечание

Следует иметь в виду, что при записи в файл, открытый с [флагом FILE\\_UNICODE](#) (либо без флага FILE\_ANSI), количество записанных байт будет в 2 раза больше количества записанных символов строки. При записи в файл, открытый с флагом FILE\_ANSI, количество записанных байт будет совпадать с количеством записанных символов строки.

### Пример:

```
//+-----+
//|                               Demo_FileWriteString.mq5 |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
```

```

//--- параметры для получения данных из терминала
input string           InpSymbolName="EURUSD";           // валютная пара
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // таймфрейм
input int               InpMAPeriod=14;                  // период скользящей средней
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;    // тип цены
input datetime          InpDateStart=D'2013.01.01 00:00'; // дата начала копирования

//--- параметры для записи данных в файл
input string           InpFileName="RSI.csv";           // имя файла
input string           InpDirectoryName="Data";          // имя директории
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    datetime date_finish; // дата конца копирования данных
    double   rsi_buff[];  // массив значений индикатора
    datetime date_buff[]; // массив дат индикатора
    int      rsi_size=0;  // размер массивов индикатора

//--- время окончания - текущее
    date_finish=TimeCurrent();
//--- получим хэндл индикатора RSI
    ResetLastError();
    int rsi_handle=iRSI(InpSymbolName, InpSymbolPeriod, InpMAPeriod, InpAppliedPrice);
    if(rsi_handle==INVALID_HANDLE)
    {
//--- не удалось получить хэндл индикатора
        PrintFormat("Ошибка получения хэндла индикатора. Код ошибки = %d", GetLastError());
        return;
    }

//--- находимся в цикле, пока индикатор не рассчитает все свои значения
    while(BarsCalculated(rsi_handle)==-1)
        Sleep(10); // задержка, чтобы индикатор успел вычислить свои значения

//--- скопируем значения индикатора за определенный период
    ResetLastError();
    if(CopyBuffer(rsi_handle,0,InpDateStart,date_finish,rsi_buff)==-1)
    {
        PrintFormat("Не удалось скопировать значения индикатора. Код ошибки = %d", GetLastError());
        return;
    }

//--- скопируем соответствующее время для значений индикатора
    ResetLastError();
    if(CopyTime(InpSymbolName, InpSymbolPeriod, InpDateStart, date_finish, date_buff)==-1)
    {
        PrintFormat("Не удалось скопировать значения времени. Код ошибки = %d", GetLastError());
        return;
    }

//--- освободим память, занимаемую индикатором
    IndicatorRelease(rsi_handle);
//--- получим размер буфера
}

```

```

rsi_size=ArraySize(rsi_buff);
//--- откроем файл для записи значений индикатора (если его нет, то создастся автоматически)
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FILE_APPEND);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("Файл %s открыт для записи",InpFileName);
    PrintFormat("Путь к файлу: %s\\Files\\\",TerminalInfoString(TERMINAL_DATA_PATH));
//--- подготовим вспомогательные переменные
string str="";
bool is_formed=false;
//--- запишем даты формирования зон перекупленности и перепроданности
for(int i=0;i<rsi_size;i++)
{
    //--- проверка значений индикатора
    if(rsi_buff[i]>=70 || rsi_buff[i]<=30)
    {
        //--- первое ли значение в этой зоне
        if(!is_formed)
        {
            //--- добавляем значение и дату
            str=(string)rsi_buff[i]+"\t"+(string)date_buff[i];
            is_formed=true;
        }
        else
            str+="\t"+(string)rsi_buff[i]+"\t"+(string)date_buff[i];
        //--- переход на следующую итерацию цикла
        continue;
    }
    //--- проверка флага
    if(is_formed)
    {
        //--- строка сформирована, запишем ее в файл
        FileWriteString(file_handle,str+"\r\n");
        is_formed=false;
    }
}
//--- закрываем файл
FileClose(file_handle);
PrintFormat("Данные записаны, файл %s закрыт",InpFileName);
}
else
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
}

```

#### Смотри также

[Тип string, StringFormat](#)

## FileWriteStruct

Записывает в бинарный файл содержимое структуры, переданной в качестве параметра, с текущего положения файлового указателя.

```
uint FileWriteStruct(
    int      file_handle,          // handle файла
    const void& struct_object,    // ссылка на объект
    int      size=-1             // размер для записи в байтах
);
```

### Параметры

*file\_handle*

[in] Файловый описатель, возвращаемый функцией [FileOpen\(\)](#).

*struct\_object*

[in] Ссылка на объект указанной структуры. Структура не должна содержать строки, [динамические массивы](#), [виртуальные функции](#), а также указатели на объекты и функции.

*size=-1*

[in] Количество байт, которые нужно записать. Если размер не указан или указано большее количество байт, чем размер структуры, то записывается вся структура полностью.

### Возвращаемое значение

В случае удачи функция возвращает количество записанных байт. Файловый указатель перемещается на это же количество байт.

### Пример:

```
//+-----+
//|                               Demo_FileWiteStruct.mq5  |
//|                               Copyright 2013, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- параметры для получения данных из терминала
input string         InpSymbolName="EURUSD";           // валютная пара
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // таймфрейм
input datetime       InpDateStart=D'2013.01.01 00:00'; // дата начала копирования данных
//--- параметры для записи данных в файл
input string         InpFileName="EURUSD.txt";          // имя файла
input string         InpDirectoryName="Data";            // имя директории
//+-----+
//| Структура для хранения данных свечи               |
//+-----+
struct candlesticks
```

```

{
    double          open;    // цена открытия
    double          close;   // цена закрытия
    double          high;    // максимальная цена
    double          low;     // минимальная цена
    datetime        date;    // дата
};

//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    datetime      date_finish=TimeCurrent();
    int           size;
    datetime      time_buff[];
    double         open_buff[];
    double         close_buff[];
    double         high_buff[];
    double         low_buff[];
    candlesticks  cand_buff[];

    //--- сбросим значение ошибки
    ResetLastError();

    //--- получим время появления баров из диапазона
    if(CopyTime(InpSymbolName, InpSymbolPeriod, InpDateStart, date_finish, time_buff)==-1)
    {
        PrintFormat("Не удалось скопировать значения времени. Код ошибки = %d", GetLastError());
        return;
    }

    //--- получим максимальные цены баров из диапазона
    if(CopyHigh(InpSymbolName, InpSymbolPeriod, InpDateStart, date_finish, high_buff)==-1)
    {
        PrintFormat("Не удалось скопировать значения максимальных цен. Код ошибки = %d", GetLastError());
        return;
    }

    //--- получим минимальные цены баров из диапазона
    if(CopyLow(InpSymbolName, InpSymbolPeriod, InpDateStart, date_finish, low_buff)==-1)
    {
        PrintFormat("Не удалось скопировать значения минимальных цен. Код ошибки = %d", GetLastError());
        return;
    }

    //--- получим цены открытия баров из диапазона
    if(CopyOpen(InpSymbolName, InpSymbolPeriod, InpDateStart, date_finish, open_buff)==-1)
    {
        PrintFormat("Не удалось скопировать значения цен открытия. Код ошибки = %d", GetLastError());
        return;
    }

    //--- получим цены закрытия баров из диапазона
    if(CopyClose(InpSymbolName, InpSymbolPeriod, InpDateStart, date_finish, close_buff)==-1)
    {
}

```

```

PrintFormat("Не удалось скопировать значения цен закрытия. Код ошибки = %d", GetLastError());
return;
}
//--- определим размерность массивов
size=ArraySize(time_buff);
//--- сохраним все данные в массиве структуры
ArrayResize(cand_buff,size);
for(int i=0;i<size;i++)
{
    cand_buff[i].open=open_buff[i];
    cand_buff[i].close=close_buff[i];
    cand_buff[i].high=high_buff[i];
    cand_buff[i].low=low_buff[i];
    cand_buff[i].date=time_buff[i];
}

//--- откроем файл для записи массива структуры в файл (если его нет, то создастся автоматически)
ResetLastError();
int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FILE_CREATE);
if(file_handle!=INVALID_HANDLE)
{
    PrintFormat("Файл %s открыт для записи",InpFileName);
    PrintFormat("Путь к файлу: %s\\Files\\\",TerminalInfoString(TERMINAL_COMMONDATA_PATH));
    //--- подготовим счетчик количества байт
    uint counter=0;
    //--- в цикле запишем значения массива
    for(int i=0;i<size;i++)
        counter+=FileWriteStruct(file_handle,cand_buff[i]);
    PrintFormat("В файл %s записано %d байт информации",InpFileName,counter);
    PrintFormat("Всего байтов: %d * %d * %d = %d, %s",size,5,8,size*5*8,size*5*8==counter?"равно":"не равно");
    //--- закрываем файл
    FileClose(file_handle);
    PrintFormat("Данные записаны, файл %s закрыт",InpFileName);
}
else
{
    PrintFormat("Не удалось открыть файл %s, Код ошибки = %d",InpFileName,GetLastError());
}
}

```

#### Смотри также

[Структуры и классы](#)

## FileLoad

Считывает всё содержимое указанного бинарного файла в переданный массив числовых типов или простых структур. Функция позволяет быстро прочитать данные известного типа в соответствующий массив.

```
long FileLoad(
    const string file_name,           // имя файла
    void& buffer[],                 // массив числовых типов или простых структур
    int common_flag=0                // файловый флаг, по умолчанию файл ищется в папке
);
```

### Параметры

*file\_name*

[in] Имя файла, из которого будет производиться считывание данных.

*buffer*

[out] Массив числовых типов или [простых структур](#).

*common\_flag=0*

[in] [Файловый флаг](#), указывающий режим работы. Если параметр не указан, то файл ищется в подпапке MQL5\Files (или <каталог\_агента\_тестирования>\MQL5\Files в случае тестирования).

### Возвращаемое значение

Количество прочитанных элементов или -1 в случае неудачи.

### Примечание

Функция FileLoad() читает из файла количество байт, кратное размеру элемента массива. Например, пусть размер файла составляет 10 байт, а чтение производится в массив типа double (`sizeof(double)=8`). В этом случае будет прочитано только 8 байт, оставшиеся 2 байта с конца файла просто будут отброшены, а сама функция FileLoad() вернёт 1 (прочитан 1 элемент).

### Пример:

```
//+-----+
//|                               Demo_FileLoad.mq5 |
//|                               Copyright 2016, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input int    bars_to_save=10; // количество баров
//+-----+
```

```
//| Script program start function
//+-----+
void OnStart()
{
    string filename=_Symbol+"_rates.bin";
    MqlRates rates[];
//---
    int copied=CopyRates(_Symbol,_Period,0,bars_to_save,rates);
    if(copied!=-1)
    {
        PrintFormat(" CopyRates(%s) copied %d bars",_Symbol,copied);
        //--- запишем котировки в файл
        if(!FileSave(filename,rates,FILE_COMMON))
            PrintFormat("FileSave() failed, error=%d",GetLastError());
    }
    else
        PrintFormat("Failed CopyRates(%s), error=%d",_Symbol,GetLastError());
//--- теперь прочитаем эти котировки обратно из файла
    ArrayFree(rates);
    long count=FileLoad(filename,rates,FILE_COMMON);
    if(count!=-1)
    {
        Print("Time\tOpen\tHigh\tLow\tClose\tTick Voulme\tSpread\tReal Volume");
        for(int i=0;i<count;i++)
        {
            PrintFormat("%s\t%G\t%G\t%G\t%I64u\t%d\t%I64u",
                        TimeToString(rates[i].time,TIME_DATE|TIME_SECONDS),
                        rates[i].open,rates[i].high,rates[i].low,rates[i].close,
                        rates[i].tick_volume,rates[i].spread,rates[i].real_volume);
        }
    }
}
```

#### Смотри также

[Структуры и классы](#), [FileReadArray](#), [FileReadStruct](#), [FileSave](#)

## FileSave

Записывает в бинарный файл все элементы массива, переданного в качестве параметра. Функция позволяет быстро в одну строку записывать массивы числовых типов или простых структур.

```
bool FileSave(
    const string file_name,           // имя файла
    void& buffer[],                 // массив числовых типов или простых структур
    int common_flag=0                // файловый флаг, по умолчанию файлы пишутся в папку
);
```

### Параметры

*file\_name*

[in] Имя файла, в который будет записан массив данных.

*buffer*

[in] Массив числовых типов или [простых структур](#).

*common\_flag=0*

[in] [Файловый флаг](#), указывающий режим работы. Если параметр не указан, то файл будет записан в подпапке MQL5\Files (или <каталог\_агента\_тестирования>\MQL5\Files в случае тестирования).

### Возвращаемое значение

В случае неудачи функция возвращает false.

### Пример:

```
//+-----+
//|                               Demo_FileSave.mq5 |
//|                               Copyright 2016, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input int      ticks_to_save=1000; // количество тиков
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    string filename=_Symbol+"_ticks.bin";
    MqlTick ticks[];
//---
    int copied=CopyTicks(_Symbol,ticks,COPY_TICKS_ALL,0,ticks_to_save);
    if(copied!=-1)
    {
```

```

PrintFormat(" CopyTicks(%s) copied %d ticks", _Symbol, copied);
//--- если тиковая история синхронизирована, то код ошибки равен нулю
if(!GetLastError() == 0)
    PrintFormat("%s: Ticks are not synchronized, error=%d", _Symbol, copied, GetLastError());
//--- запишем тики в файл
if(!FileSave(filename, ticks, FILE_COMMON))
    PrintFormat("FileSave() failed, error=%d", GetLastError());
}
else
    PrintFormat("Failed CopyTicks(%s), Error=%d", _Symbol, GetLastError());
//--- теперь прочитаем эти тики обратно из файла
ArrayFree(ticks);
long count=FileLoad(filename, ticks, FILE_COMMON);
if(count != -1)
{
    Print("Time\tBid\tAsk\tLast\tVolume\tms\tFlags");
    for(int i=0;i<count;i++)
    {
        PrintFormat("%s.%03I64u:\t%G\t%G\t%G\t%I64u\t0x%04x",
        TimeToString(ticks[i].time, TIME_DATE | TIME_SECONDS), ticks[i].time_msc % 1000,
        ticks[i].bid, ticks[i].ask, ticks[i].last, ticks[i].volume, ticks[i].flags);
    }
}
}
}

```

#### Смотри также

[Структуры и классы](#), [FileWriteArray](#), [FileWriteStruct](#), [FileLoad](#), [FileWrite](#)

## FolderCreate

Создает директорию в директории Files (в зависимости от значения common\_flag)

```
bool FolderCreate(
    string folder_name,           // строка с именем создаваемой папки
    int     common_flag=0         // область действия
);
```

### Параметры

*folder\_name*

[in] Имя директории, которую требуется создать. Содержит относительный путь к папке.

*common\_flag=0*

[in] [Флаг](#), определяющий местоположение директории. Если common\_flag=FILE\_COMMON, то директория находится в общей папке всех клиентских терминалов \Terminal\Common\Files. В противном случае директория находится в локальной папке (MQL5\Files или MQL5\Tester\Files в случае тестирования).

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Из соображений безопасности в языке MQL5 строго контролируется работа с файлами. Файлы, с которыми проводятся файловые операции средствами языка MQL5, не могут находиться за пределами файловой "песочницы".

### Пример:

```
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- описание
#property description "Скрипт показывает пример использования FolderCreate()."
#property description "Внешний параметр определяет папку для создания папок."
#property description "После выполнения скрипта будет создана структура папок"

//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входной параметр определяет папку, в которой работает скрипт
input bool     common_folder=false; // общая папка всех терминалов
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- папка, которую создадим в MQL5\Files
    string root_folder="Folder_A";
    if(CreateFolder(root_folder,common_folder))
```

```

{
    //--- создадим в ней дочернюю папку Child_Folder_B1
    string folder_B1="Child_Folder_B1";
    string path=root_folder+"\\"+folder_B1;           // создадим имя папки с учетом
    if(CreateFolder(path,common_folder))
    {
        //--- в этой папке создадим еще 3 дочерних
        string folder_C11="Child_Folder_C11";
        string child_path=root_folder+"\\"+folder_C11;// создадим имя папки с учетом
        CreateFolder(child_path,common_folder);
        //--- вторая дочерняя папка
        string folder_C12="Child_Folder_C12";
        child_path=root_folder+"\\"+folder_C12;
        CreateFolder(child_path,common_folder);

        //--- третья дочерняя папка
        string folder_C13="Child_Folder_C13";
        child_path=root_folder+"\\"+folder_C13;
        CreateFolder(child_path,common_folder);
    }
}
//---
//+-----+
// | Пытается создать папку и выводит сообщение об этом |
//+-----+
bool CreateFolder(string folder_path,bool common_flag)
{
    int flag=common_flag?FILE_COMMON:0;
    string working_folder;
    //--- выясним полный путь в зависимости от параметра common_flag
    if(common_flag)
        working_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH)+"\\MQL5\\Files";
    else
        working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL5\\Files";
    //--- отладочное сообщение
    PrintFormat("folder_path=%s",folder_path);
    //--- попытка создать папку относительно пути MQL5\Files
    if(FolderCreate(folder_path,flag))
    {
        //--- выведем полный путь для созданной папки
        PrintFormat("Создали папку %s",working_folder+"\\"+folder_path);
        //--- сбросим код ошибки
        ResetLastError();
        //--- успешное выполнение
        return true;
    }
    else
        PrintFormat("Не удалось создать папку %s. Код ошибки %d",working_folder+folder_
}

```

```
//--- неудачное выполнение  
    return false;  
}
```

Смотри также

[FileOpen\(\)](#), [FolderClean\(\)](#), [FileCopy\(\)](#)

## FolderDelete

Удаляет указанную директорию. Если папка не пуста, то она не может быть удалена.

```
bool FolderDelete(
    string folder_name,           // строка с именем удаляемой папки
    int common_flag=0             // область действия
);
```

### Параметры

*folder\_name*

[in] Имя директории, которую требуется удалить. Содержит полный путь к папке.

*common\_flag=0*

[in] [Флаг](#), определяющий местоположение директории. Если *common\_flag*=FILE\_COMMON, то директория находится в общей папке всех клиентских терминалов \Terminal\Common\Files. В противном случае директория находится в локальной папке (MQL5\Files или MQL5\Tester\Files в случае тестирования).

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Из соображений безопасности в языке MQL5 строго контролируется работа с файлами. Файлы, с которыми проводятся файловые операции средствами языка MQL5, не могут находиться за пределами файловой "песочницы".

Если директория содержит хоть один файл и/или поддиректорию, то удаление такой директории невозможно, ее необходимо предварительно очистить. Тотальная очистка папки от всех файлов и всех вложенных подпапок осуществляется при помощи функции [FolderClean\(\)](#).

### Пример:

```
//+-----+
//|                               Demo_FolderDelete.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- описание
#property description "Скрипт показывает пример использования FolderDelete()."
#property description "Сначала создаются две папки, одна пустая, другая содержит файл."
#property description "При попытке удаления непустой папки получим ошибку и предупреждение."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input string   firstFolder="empty";      // пустая папка
```

```

input string    secondFolder="nonempty"; // папка, в которой будет один файл
string filename="delete_me.txt";           // имя файла, который мы создадим в папке secondFolder
//-----
//| Script program start function
//+-----+
void OnStart()
{
//--- хендл файла запишем сюда
int handle;
//--- выясним в какой папке мы работаем
string working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL5\\Files";
//--- отладочное сообщение
PrintFormat("working_folder=%s",working_folder);
//--- попытка создать пустую папку относительно пути MQL5\Files
if(FolderCreate(firstFolder,0)) // 0 означает, что работаем в локальной папке терминала
{
//--- выведем полный путь до созданной папки
PrintFormat("Создали папку %s",working_folder+"\\"+firstFolder);
//--- сбросим код ошибки
ResetLastError();
}
else
PrintFormat("Не удалось создать папку %s. Код ошибки %d",working_folder+"\\"+firstFolder,GetLastError());
//--- теперь создадим непустую папку с помощью функции FileOpen()
string filepath=secondFolder+"\\"+filename; // сформируем путь для файла, который хотим удалить
handle=FileOpen(filepath,FILE_WRITE|FILE_TXT); // флаг FILE_WRITE в данном случае означает запись
if(handle!=INVALID_HANDLE)
PrintFormat("Открыли файл на чтение %s",working_folder+"\\"+filepath);
else
PrintFormat("Не удалось создать файл %s в папке %s. Код ошибки=%d",filename,secondFolder,GetLastError());
Comment(StringFormat("Готовимся удалить папки %s и %s", firstFolder, secondFolder));
//--- Небольшая пауза в 5 секунд, чтобы мы могли прочитать сообщение на графике
Sleep(5000); // Sleep() нельзя использовать в индикаторах!
//--- выведем диалоговое окно и просим пользователя
int choice=MessageBox(StringFormat("Удалить папки %s и %s?", firstFolder, secondFolder),
"Удаление папок",
MB_YESNO|MB_ICONQUESTION); // будут две кнопки - "Yes" и "No"
//--- выполним действия в зависимости от выбранного варианта
if(choice==IDYES)
{
//--- очистим комментарий на графике
Comment("");
//--- выведем сообщение в журнал "Эксперты"
PrintFormat("Пробуем удалить папки %s и %s",firstFolder, secondFolder);
ResetLastError();
//--- удаляем пустую папку
}
}

```

```
if(FolderDelete(firstFolder))
    //--- должны увидеть это сообщение, так как папка пустая
    PrintFormat("Папка %s успешно удалена", firstFolder);
else
    PrintFormat("Не удалось удалить папку %s. Код ошибки=%d", firstFolder, GetLastError());
ResetLastError();
//--- удаляем папку, которая содержит файл
if(FolderDelete(secondFolder))
    PrintFormat("Папка %s успешно удалена", secondFolder);
else
    //--- должны увидеть это сообщение, так как в папке есть файл
    PrintFormat("Не удалось удалить папку %s. Код ошибки=%d", secondFolder, GetLastError());
}
else
    Print("Удаление отменено");
//---
}
```

#### Смотри также

[FileOpen\(\)](#), [FolderClean\(\)](#), [FileMove\(\)](#)

## FolderClean

Удаляет все файлы в указанной папке.

```
bool FolderClean(
    string folder_name,           // строка с именем папки
    int common_flag=0             // область действия
);
```

### Параметры

*folder\_name*

[in] Имя директории, в которой требуется удалить все файлы. Содержит полный путь к папке.

*common\_flag=0*

[in] Флаг, определяющий местоположение директории. Если *common\_flag*=FILE\_COMMON, то директория находится в общей папке всех клиентских терминалов \Terminal\Common\Files. В противном случае директория находится в локальной папке (MQL5\Files или MQL5\Tester\Files в случае тестирования).

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Из соображений безопасности в языке MQL5 строго контролируется работа с файлами. Файлы, с которыми проводятся файловые операции средствами языка MQL5, не могут находиться за пределами "файловой песочницы".

Осторожно пользуйтесь этой функцией, так как все файлы и все вложенные поддиректории удаляются безвозвратно.

### Пример:

```
//+-----+
//|                               Demo_FolderClean.mq5 |
//|           Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- описание
#property description "Скрипт показывает пример использования FolderClean()."
#property description "Сначала создаются файлы в указанной папке с помощью функции FileCreate."
#property description "Затем перед удалением файлов выводится предупреждение MessageBox."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры
input string   foldername="demo_folder"; // создаем папку в MQL5/Files/
input int      files=5;                  // сколько файлов создадим и удалим
```

```

//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    string name="testfile";
//--- сначала откроем или создадим файлы в папке данных нашего терминала
for(int N=0;N<files;N++)
{
    //--- соберем имя файла в виде 'demo_folder\testfileN.txt'
    string filemane=StringFormat("%s\\%s%d.txt",foldername,name,N);
    //--- открываем файл с флагом на запись, в этом случае папка 'demo_folder' будет
    int handle=FileOpen(filemane,FILE_WRITE);
    //--- выясним, насколько успешно отработала функция FileOpen()
    if(handle==INVALID_HANDLE)
    {
        PrintFormat("Не удалось создать файл %s. Код ошибки %d",filemane,GetLastError());
        ResetLastError();
    }
    else
    {
        PrintFormat("Файл %s успешно открыт",filemane);
        //--- открытый файл нам больше не нужен, обязательно закрываем
        FileClose(handle);
    }
}

//--- проверим, сколько файлов в папке
int k=FilesInFolder(foldername+"\*.*",0);
PrintFormat("Всего в папке %s найдено %d файлов",foldername,k);

//--- выведем диалоговое окно и спросим пользователя
int choice=MessageBox(StringFormat("Вы собираетесь удалить из папки %s %d файлов, т",
                                    "Удаление файлов из папки",
                                    MB_YESNO|MB_ICONQUESTION); // будут две кнопки - "Yes" и "No"
ResetLastError();

//--- выполним действия в зависимости от выбранного варианта
if(choice==IDYES)
{
    //--- начинаем удалять
    PrintFormat("Попытка удалить все файлы из папки %s",foldername);
    if(FolderClean(foldername,0))
        PrintFormat("Файлы успешно удалены, в папке %s осталось %d файлов",
                    foldername,
                    FilesInFolder(foldername+"\*.*",0));
    else
        PrintFormat("Не удалось удалить файлы из папки %s. Код ошибки %d",foldername,
}
else

```

```
PrintFormat("Удаление отменено");

//---
}

//+-----+
//| возвращает количество файлов в указанной папке |
//+-----+

int FilesInFolder(string path,int flag)
{
    int count=0;
    long handle;
    string filename;
//---

    handle=FileFindFirst(path,filename,flag);
//--- если хотя бы один файл найден, ищем остальные
    if(handle!=INVALID_HANDLE)
    {
//--- выведем имя файла
        PrintFormat("найден файл %s",filename);
//--- увеличим счетчик найденных файлов/папок
        count++;
//--- начинаем перебор всех файлов/папок
        while(FileFindNext(handle,filename))
        {
            PrintFormat("найден файл %s",filename);
            count++;
        }
//--- обязательно закрываем хендл поиска по окончании
        FileFindClose(handle);
    }
    else // хендл получить не удалось
    {
        PrintFormat("Не удалось провести поиск файлов в папке %s",path);
    }
//--- вернем результат
    return count;
}
```

#### Смотри также

[FileFindFirst](#), [FileFindNext](#), [FileFindClose](#)

## Пользовательские индикаторы

Группа функций, используемых при оформлении пользовательских индикаторов. Данные функции нельзя использовать при написании советников и скриптов.

Функция	Действие
<a href="#">SetIndexBuffer</a>	Связывает указанный индикаторный буфер с одномерным динамическим <a href="#">массивом</a> типа <a href="#">double</a>
<a href="#">IndicatorSetDouble</a>	Задает значение свойства индикатора, имеющего тип <a href="#">double</a>
<a href="#">IndicatorSetInteger</a>	Задает значение свойства индикатора, имеющего тип <a href="#">int</a>
<a href="#">IndicatorSetString</a>	Задает значение свойства индикатора, имеющего тип <a href="#">string</a>
<a href="#">PlotIndexSetDouble</a>	Задает значение свойства линии индикатора, имеющего тип <a href="#">double</a>
<a href="#">PlotIndexSetInteger</a>	Задает значение свойства линии индикатора, имеющего тип <a href="#">int</a>
<a href="#">PlotIndexSetString</a>	Задает значение свойства линии индикатора, имеющего тип <a href="#">string</a>
<a href="#">PlotIndexGetInteger</a>	Возвращает значение свойства линии индикатора, имеющего <a href="#">целый</a> тип

[Свойства индикаторов](#) можно устанавливать как с помощью директив компилятора, так и с помощью функций. Для лучшего понимания рекомендуется изучить [стили индикаторов в примерах](#).

Все необходимые расчеты пользовательских индикаторов необходимо размещать в предопределенной функции [OnCalculate\(\)](#). Если используется короткая форма вызова функции OnCalculate() вида

```
int OnCalculate (const int rates_total, const int prev_calculated, const int begin, const int end)
```

то переменная *rates\_total* содержит значение общего количества элементов массива *price[]*, переданного в качестве входного параметра для расчета значений индикатора.

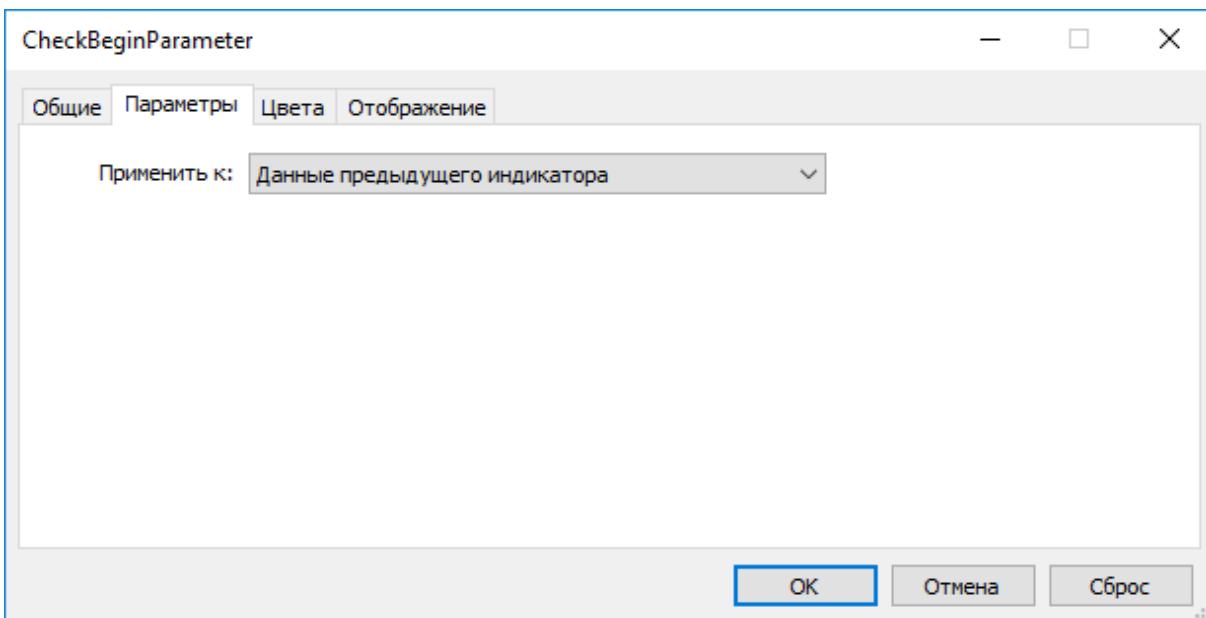
Параметр *prev\_calculated* - результат выполнения функции OnCalculate() на предыдущем вызове и позволяет организовать экономный алгоритм расчета значений индикатора. Например, если текущее значение *rates\_total=1000*, а *prev\_calculated=999*, то, возможно, нам достаточно сделать расчеты только для одного значения каждого индикаторного буфера.

Если бы информация о размере входного массива *price* была бы недоступна, то это привело бы к необходимости производить расчеты для 1000 значений каждого индикаторного буфера. При первом вызове функции OnCalculate() значение *prev\_calculated=0*. Если массив *price[]* каким-либо образом изменился, то в этом случае *prev\_calculated* также равно 0.

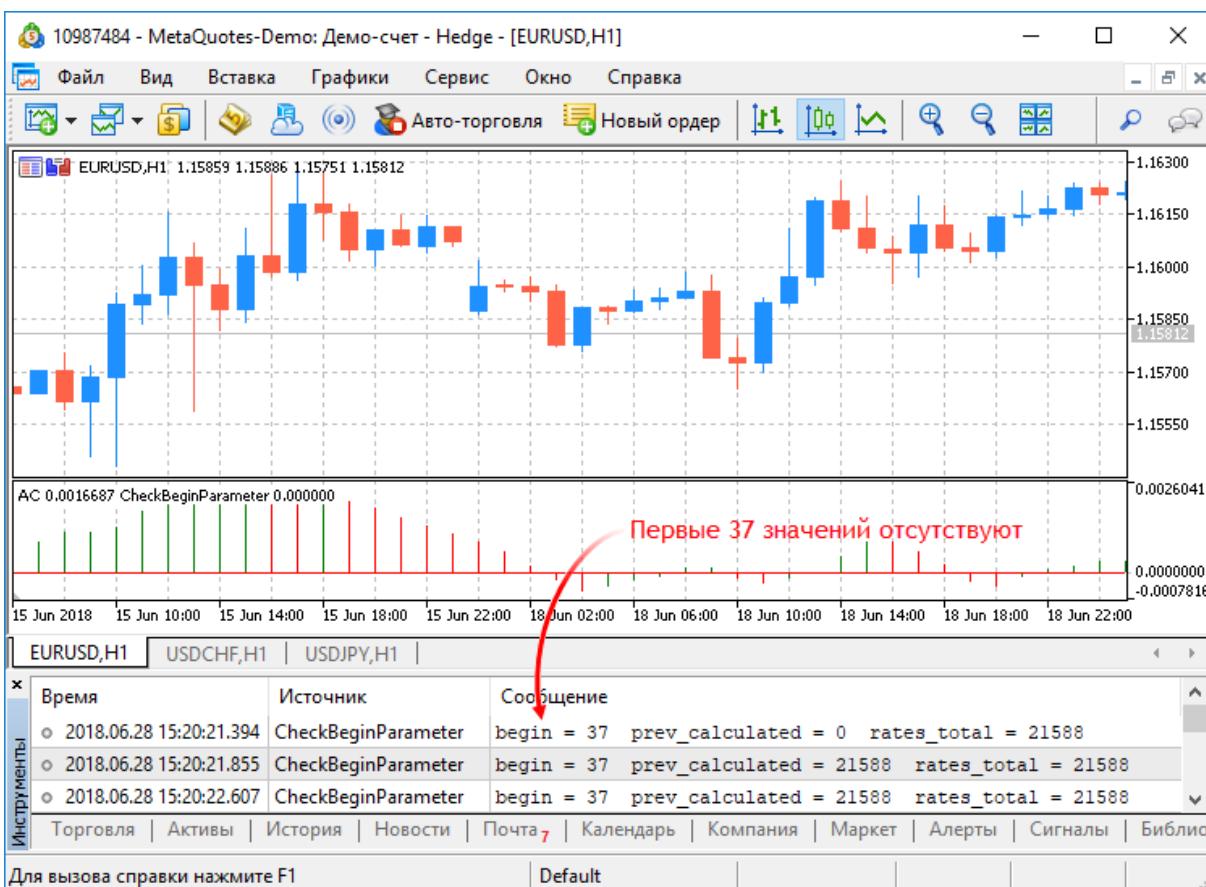
Параметр `begin` сообщает количество начальных значений массива `price`, которые не содержат данных для расчета. Например, если в качестве входного массива были использованы значения индикатора Accelerator Oscillator (для которого первые 37 значений не рассчитываются), то `begin=37`. Для примера рассмотрим простой индикатор:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- plot Label1
#property indicator_label1 "Label1"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- indicator buffers
double Label1Buffer[];
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
//---
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double &price[])
{
//---
Print("begin = ",begin," prev_calculated = ",prev_calculated," rates_total = ",rates_total," price[0] = ",price[0]);
//--- return value of prev_calculated for next call
return(rates_total);
}
```

Перетащим его из окна "Навигатора" на окно индикатора Accelerator Oscillator и укажем, что расчеты будут производиться на значениях предыдущего индикатора:



В результате при первом вызове функции `OnCalculate()` значение `prev_calculated` окажется равным нулю, а при последующих вызовах оно будет равно значению `rates_total` (до тех пор, пока не увеличится количество баров на ценовом графике).



Значение параметра `begin` будет в точности равно количеству начальных баров, для которых значения индикатора Accelerator не рассчитывается в соответствие с логикой этого индикатора. Если мы посмотрим исходный код пользовательского индикатора `Accelerator.mq5`, то увидим в функции `OnInit()` такие строчки:

```
//--- sets first bar from what index will be drawn  
PlotIndexSetInteger(0, PLOT_DRAW_BEGIN, 37);
```

Именно функцией [PlotIndexSetInteger\(0, PLOT\\_DRAW\\_BEGIN, empty\\_first\\_values\)](#) мы сообщаем количество несущественных первых значений в нулевом индикаторном массиве пользовательского индикатора, которые нам не нужно принимать для расчетов (`empty_first_values`). Таким образом, у нас есть механизмы, чтобы:

1. сообщить о количестве начальных значений индикатора, которые не стоит использовать для расчетов в другом пользовательском индикаторе;
2. получить информацию о количестве первых значений, которые необходимо игнорировать при вызове другого пользовательского индикатора, не вдаваясь в логику его расчетов.

## Стили индикаторов в примерах

В клиентский терминал MetaTrader 5 встроено 38 технических индикаторов, которые можно использовать в программах MQL5 с помощью [соответствующих функций](#). Но главное достоинство языка MQL5 - возможность создавать свои собственные пользовательские индикаторы, которые потом можно использовать в советниках для получения значений или просто накладывать на ценовые графики для проведения технического анализа.

Все множество индикаторов можно получить на основе нескольких базовых [стилей рисования](#), называемых графическими построениями. Под построением понимается способ отображения данных, которые индикатор рассчитывает, хранит и выдает по запросу. Всего таких базовых построений семь:

1. линия,
2. секция (отрезок),
3. гистограмма,
4. стрелка (символ),
5. закрашиваемая область (канал с заливкой),
6. бары,
7. японские свечи.

Каждое построение требует для своего отображения от одного до пяти [массивов](#) типа [double](#), в которых хранятся значения индикатора. Эти массивы для удобства работы индикатора связываются с индикаторными буферами. Количество буферов в индикаторе необходимо объявить заранее с помощью [директивы компилятора](#), пример:

```
#property indicator_buffers 3 // количество буферов
#property indicator_plots 2 // количество графических построений
```

Количество буферов в индикаторе всегда больше или равно количеству построений в индикаторе.

Так как каждое базовое графическое построение может иметь цветовые вариации или специфику отображения, то реальное количество построений в языке MQL5 составляет 18:

Построение	Описание	Буферов значений	Буферов цвета
<a href="#">DRAW_NONE</a>	На графике визуально не отображается, но значения соответствующего буфера можно посмотреть в "Окне данных"	1	-
<a href="#">DRAW_LINE</a>	Строится линия по значениям соответствующего буфера (пустые значения в буфере нежелательны)	1	-

<u>DRAW_SECTION</u>	Рисуется отрезками между значениями соответствующего буфера (обычно пустых значений много)	1	-
<u>DRAW_HISTOGRAM</u>	Рисуется гистограммой от нулевой линии до значений соответствующего буфера (допускаются пустые значения)	1	-
<u>DRAW_HISTOGRAM2</u>	Рисуется гистограммой на двух индикаторных буферах (допускаются пустые значения)	2	-
<u>DRAW_ARROW</u>	Отображается символами (допускаются пустые значения)	1	-
<u>DRAW_ZIGZAG</u>	Похож на стиль <u>DRAW_SECTION</u> , но в отличие от него может строить вертикальные отрезки на одном баре	2	-
<u>DRAW_FILLING</u>	Цветовая заливка между двумя линиями. В "Окне данных" показываются 2 значения соответствующих буферов	2	-
<u>DRAW_BARS</u>	Отображение на графике в виде баров. В "Окне данных" показываются 4 значения соответствующих буферов	4	-

<a href="#"><u>DRAW_CANDLES</u></a>	Отображение в виде японских свечей. В "Окне данных" показываются 4 значения соответствующих буферов	4	-
<a href="#"><u>DRAW_COLOR_LINE</u></a>	Линия, для которой можно чередовать цвет как на разных барах, так и сменить ее цвет в любой момент времени	1	1
<a href="#"><u>DRAW_COLOR_SECTION</u></a>	Похож на стиль <a href="#"><u>DRAW_SECTION</u></a> , но цвет каждой секции можно задавать индивидуально, можно задавать цвет динамически	1	1
<a href="#"><u>DRAW_COLOR_HISTOGRAM</u></a>	Похож на стиль <a href="#"><u>DRAW_HISTOGRAM</u></a> , но каждая полоска может иметь свой цвет, цвет можно задавать цвет динамически	1	1
<a href="#"><u>DRAW_COLOR_HISTOGRAM2</u></a>	Похож на <a href="#"><u>DRAW_HISTOGRAM2</u></a> , но каждая полоска может иметь свой цвет, цвет можно задавать цвет динамически	2	1
<a href="#"><u>DRAW_COLOR_ARROW</u></a>	Похож на стиль <a href="#"><u>DRAW_ARROW</u></a> , но каждый символ может иметь свой цвет. Цвет можно менять динамически	1	1
<a href="#"><u>DRAW_COLOR_ZIGZAG</u></a>	Стиль <a href="#"><u>DRAW_ZIGZAG</u></a> с возможностями индивидуальной раскраски секций и динамической смены цвета	2	1

<a href="#">DRAW_COLOR_BARS</a>	Стиль <a href="#">DRAW_BARS</a> с возможностями индивидуальной раскраски баров и динамической смены цвета	4	1
<a href="#">DRAW_COLOR_CANDLES</a>	Стиль <a href="#">DRAW_CANDLES</a> с возможностями индивидуальной раскраски свечей и динамической смены цвета	4	1

## Разница между индикаторным буфером и массивом

В каждом индикаторе необходимо объявить на [глобальном уровне](#) один или более массивов типа `double`, который затем должен быть использован в качестве индикаторного буфера с помощью функции [SetIndexBuffer\(\)](#). Для рисования графических построений индикатора используются только значения из индикаторных буферов, какие-либо другие массивы для этого использовать нельзя. Кроме того, значения буферов показываются в "Окне данных".

Индикаторный буфер должен быть [динамическим](#) и не требует [указания размера](#) - размер массива, использованного в качестве индикаторного буфера, устанавливается исполняющей подсистемой терминала автоматически.

[Направление индексации](#) после связывания массива с индикаторным буфером по умолчанию устанавливается как в обычных массивах, но при необходимости можно применить функцию [ArraySetAsSeries\(\)](#) для изменения способа доступа к элементам массива. По умолчанию индикаторный буфер используется для хранения данных, предназначенных для отрисовки ([INDICATOR\\_DATA](#)).

Если для расчетов значений индикатора требуется проводить промежуточные вычисления и хранить для каждого бара вспомогательное значение, то при связывании такой массив можно объявить в качестве расчетного буфера ([INDICATOR\\_CALCULATIONS](#)). Можно использовать для промежуточных значений и обычный массив, но в этом случае программист должен самостоятельно управлять размером такого массива.

Некоторые построения позволяют задавать для каждого бара цвет отображения. Для хранения информации о цвете используются цветовые буфера ([INDICATOR\\_COLOR\\_INDEX](#)). Цвет представлен целочисленным типом `color`, но все индикаторные буфера должны иметь тип `double`. Значения цветовых и вспомогательных ([INDICATOR\\_CALCULATIONS](#)) буферов нельзя получить с помощью функции [CopyBuffer\(\)](#).

Количество индикаторных буферов должно быть указано директивой компилятора `#property indicator_buffers` количество\_буферов:

```
#property indicator_buffers 3 // индикатор имеет 3 буфера
```

Максимально допустимое количество буферов в одном индикаторе - 512.

## Соответствие индикаторных буферов и графических построений

Каждое графическое построение базируется на одном или более индикаторных буферах. Так, для отображения простых японских свечей требуется четыре значения - цены Open, High, Low и Close. Соответственно, для отображения индикатора в виде японских свечей необходимо объявить 4 индикаторных буфера и 4 массива типа double под них. Например:

```
//--- в индикаторе четыре индикаторных буфера
#property indicator_buffers 4
//--- в индикаторе одно графическое построение
#property indicator_plots 1
//--- графическое построение под номером 1 будет отображаться японскими свечами
#property indicator_type1 DRAW_CANDLES
//--- японские свечи будут рисоваться цветом clrDodgerBlue
#property indicator_color1 clrDodgerBlue
//--- 4 массива под индикаторные буфера
double OBuffer[];
double HBuffer[];
double LBuffer[];
double CBuffer[];
```

Графические построения автоматически используют индикаторные буфера в соответствии с номером построения. Номера построения начинаются с единицы, номера буферов начинаются с нуля. Если первое построение требует 4 индикаторных буфера, то для отрисовки будут использованы 4 первых индикаторных буфера. Эти четыре буфера должны быть связаны функцией [SetIndexBuffer\(\)](#) с соответствующими массивами с правильной индексацией.

```
//--- связывание массивов с индикаторными буферами
SetIndexBuffer(0,OBuffer,INDICATOR_DATA); // первый буфер соответствует нулевому построению
SetIndexBuffer(1,HBuffer,INDICATOR_DATA); // второй буфер соответствует индексу 1
SetIndexBuffer(2,LBuffer,INDICATOR_DATA); // третий буфер соответствует индексу 2
SetIndexBuffer(3,CBuffer,INDICATOR_DATA); // четвертый буфер соответствует индексу 3
```

При отрисовке японских свечей индикатор будет использовать именно первые четыре буфера, потому что построение "японские свечи" было объявлено под первым номером.

Изменим немного пример, добавим построение в виде простой линии - [DRAW\\_LINE](#). Пусть теперь линия будет иметь номер 1, а японские свечи будут под номером 2. Количество буферов и количество построений увеличилось.

```
//--- в индикаторе 5 индикаторных буферов
#property indicator_buffers 5
//--- в индикаторе 2 графических построения
#property indicator_plots 2
//--- графическое построение под номером 1 будет отображаться линией
#property indicator_type1 DRAW_LINE
//--- линия будет рисоваться цветом clrDodgerRed
#property indicator_color1 clrDodgerRed
```

```

//--- графическое построение под номером 2 будет отображаться японскими свечами
#property indicator_type2 DRAW_CANDLES
//--- японские свечи будут рисоваться цветом clrDodgerBlue
#property indicator_color2 clrDodgerBlue
//--- 5 массивов под индикаторные буферы
double LineBuffer[];
double OBuffer[];
double HBuffer[];
double LBuffer[];
double CBuffer[];

```

Порядок следования построений изменился, теперь первым идет линия, следом японские свечи. Поэтому и порядок следования буферов будет таким же - сначала объявим под нулевым индексом буфер для линии, а затем четыре буфера для отображения японских свечей.

```

SetIndexBuffer(0,LineBuffer,INDICATOR_DATA); // первый буфер соответствует индексу
//--- связывание массивов с индикаторными буферами под японские свечи
SetIndexBuffer(1,OBuffer,INDICATOR_DATA); // второй буфер соответствует индексу
SetIndexBuffer(2,HBuffer,INDICATOR_DATA); // третий буфер соответствует индексу
SetIndexBuffer(3,LBuffer,INDICATOR_DATA); // четвертый буфер соответствует индексу
SetIndexBuffer(4,CBuffer,INDICATOR_DATA); // пятый буфер соответствует индексу

```

Количество буферов и графических построений можно задавать только с помощью директив компилятора, динамическое изменение этих свойств с помощью функций невозможно.

## Цветовые версии стилей

Как можно увидеть в таблице, стили делятся на две группы. Первая группа – это стили, у которых в названии нет слова **COLOR**, назовем эти стили базовыми:

- DRAW\_LINE
- DRAW\_SECTION
- DRAW\_HISTOGRAM
- DRAW\_HISTOGRAM2
- DRAW\_ARROW
- DRAW\_ZIGZAG
- DRAW\_FILLING
- DRAW\_BARS
- DRAW\_CANDLES

Вторая группа стилей содержит в названии слово **COLOR**, назовем их цветовыми версиями:

- DRAW\_COLOR\_LINE
- DRAW\_COLOR\_SECTION
- DRAW\_COLOR\_HISTOGRAM
- DRAW\_COLOR\_HISTOGRAM2
- DRAW\_COLOR\_ARROW

- DRAW\_COLOR\_ZIGZAG
- DRAW\_COLOR\_BARS
- DRAW\_COLOR\_CANDLES

Все цветовые версии стилей отличаются от базовых тем, что позволяют задавать цвет для каждой части графического построения. Минимальной частью построения является бар, поэтому можно сказать, что цветовые версии позволяют задавать цвет построения на каждом баре.

Для того чтобы задавать цвет построения на каждом баре, в цветовые версии стилей был добавлен дополнительный специальный буфер для хранения индекса цвета. Эти индексы указывают на номер цвета в специальном массиве, содержащем заранее определенный набор цветов. Размер массива цветов - 64. Это означает, что каждая цветовая версия стиля позволяет раскрасить графическое построение 64-мя различными цветами.

Набор и количество цветов в специальном массиве цветов можно задавать директивой компилятора #property indicator\_color, где через запятую указываются все необходимые цвета. Например, такая запись в индикаторе:

```
//--- зададим 8 цветов для раскраски свечей (они хранятся в специальном массиве)
#property indicator_color1 clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL:
```

Здесь указано, что для графического построения номер 1 заданы 8 цветов, которые будут помещены в специальный массив. Далее в программе мы будем указывать не сам цвет, которым будет отображаться графическое построение, а только его индекс. Если мы хотим задать для бара красный цвет, то для этого необходимо установить в цветовом буфере индекс красного цвета из массива. Красный цвет задан в директиве первым первым, ему соответствует индекс номер 0.

```
//--- зададим цвет свечи clrRed
col_buffer[buffer_index]=0;
```

Набор цветов для раскрашивания не является раз и навсегда заданным, его можно менять динамически с помощью функции PlotIndexSetInteger(). Пример:

```
//--- установим цвет для каждого индекса как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0, // номер графического стиля
                     PLOT_LINE_COLOR, // идентификатор свойства
                     plot_color_ind, // индекс цвета, куда запишем цвет
                     color_array[i]); // новый цвет
```

## Свойства индикатора и графических построений

Для графических построений свойства можно устанавливать как с помощью [директив компилятора](#), так и с помощью соответствующих функций. Подробней об этом написано в разделе [Связь между свойствами индикатора и функциями](#). Динамическое изменение свойств индикатора с помощью функций позволяет создавать более гибкие пользовательские индикаторы.

## Начало отрисовки индикатора на графике

Во многих случаях по условиям алгоритма расчет значений индикатора невозможно начать сразу же с текущего бара, требуется обеспечить минимальное количество предыдущих доступных баров в истории. Например, многие виды сглаживания подразумевают, что берется массив цен за предыдущие N баров, и на основании этих значений рассчитывается значение индикатора для текущего бара.

В таких случаях либо нет возможности рассчитать значения индикатора на N самых первых барах, либо эти значения не предназначены для отображения на графике и являются только вспомогательными для расчета последующих значений. Чтобы отказаться от визуализации индикатора на первых N барах истории, следует установить свойству [PLOT\\_DRAW\\_BEGIN](#) значение N для соответствующего графического построения:

```
//--- связывание массивов с индикаторными буферами под японские свечи  
PlotIndexSetInteger(номер_графического_построения, PLOT_DRAW_BEGIN, N);
```

Здесь:

- номер\_графического\_построения - значение от нуля до indicator\_plots-1 (нумерация графических построений начинается с нуля).
- N - количество первых в истории баров, на которых индикатор не должен отображаться на графике.

## DRAW\_NONE

Стиль DRAW\_NONE предназначен для тех случаев, когда значения буфера необходимо рассчитывать и показывать в "Окне данных", но само отображение на графике не требуется. Для настройки точности отображения используйте в функции [OnInit\(\)](#) выражение `IndicatorSetInteger(INICATOR_DIGITS, количество_знаков)`:

```
int OnInit()
{
    //--- indicator buffers mapping
    SetIndexBuffer(0,InvisibleBuffer,INDICATOR_DATA);
    //--- установим точность, с которой значение будет показываться в Окне данных
    IndicatorSetInteger(INICATOR_DIGITS,0);
    //---
    return(INIT_SUCCEEDED);
}
```

Количество требуемых буферов для построения DRAW\_NONE – 1.

Пример индикатора, который показывает в "Окне данных" номер бара, на котором находится мышка. Нумерация соответствует таймсерии, то есть текущий незавершенный бар имеет нулевой индекс, а самый старый бар имеет самый большой индекс.



Обратите внимание, что несмотря на то, что для графического построения №1 указан красный цвет отображения, индикатор ничего не рисует на графике.

```
//+-----+
//|                               DRAW_NONE.mq5  |
//| Copyright 2011, MetaQuotes Software Corp.  |
//| https://www.mql5.com  |
```

```

//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Invisible
#property indicator_label1 "Bar Index"
#property indicator_type1  DRAW_NONE
#property indicator_style1 STYLE_SOLID
#property indicator_color1 clrRed
#property indicator_width1 1
//--- indicator buffers
double      InvisibleBuffer[];
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
    //--- связывание массива и индикаторного буфера
    SetIndexBuffer(0,InvisibleBuffer,INDICATOR_DATA);
    //--- установим точность, с которой значение будет показываться в "Окне данных"
    IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function               |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static datetime lastbar=0;
//--- если это первый расчет индикатора
    if(prev_calculated==0)
    {
        //--- перенумеруем бары в первый раз
        CalcValues(rates_total,close);
        //--- запомним время открытия текущего бара в lastbar
        lastbar=(datetime)SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE);
    }
}

```

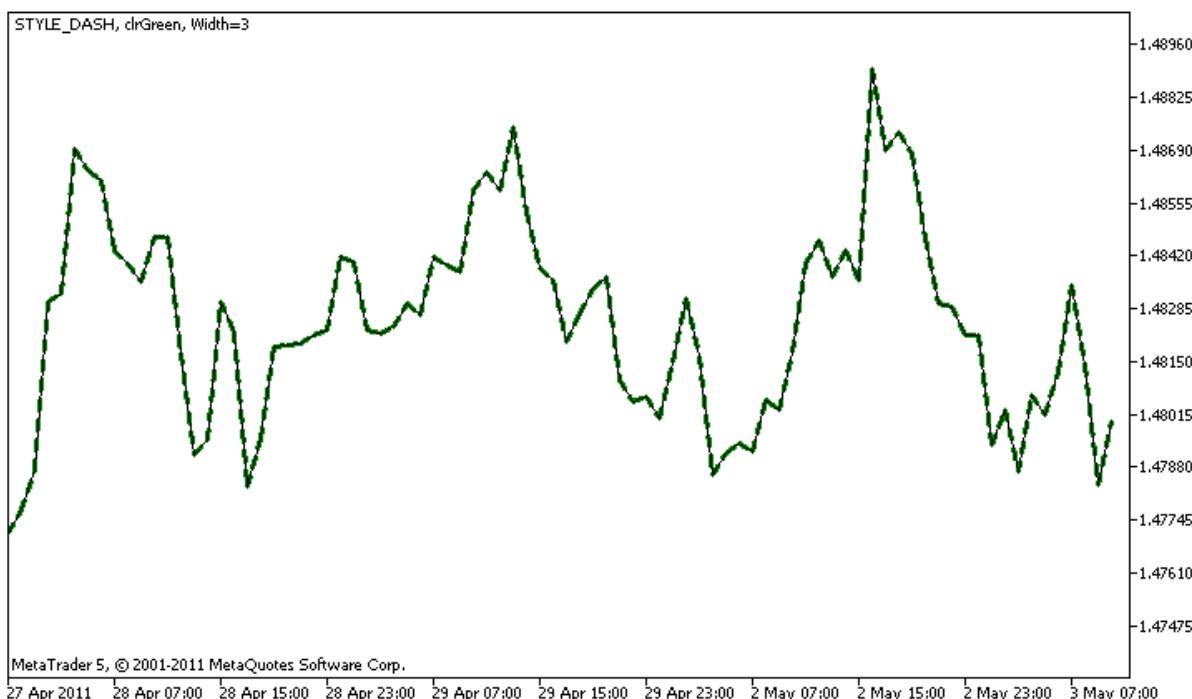
```
        }
    else
    {
        //--- если появился новый бар, время его открытия не совпадает с lastbar
        if(lastbar!=SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE))
        {
            //--- перенумеруем бары заново
            CalcValues(rates_total,close);
            //--- обновим время открытия текущего бара в lastbar
            lastbar=(datetime)SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE);
        }
    }
    //--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+
//| Нумерует бары как в таймсерии |
//+-----+
void CalcValues(int total,double const &array[])
{
    //--- зададим индикаторному буферу индексацию как в таймсерии
    ArraySetAsSeries(InvisibleBuffer,true);
    //--- заполним каждому бару его номер
    for(int i=0;i<total;i++) InvisibleBuffer[i]=i;
}
```

## DRAW\_LINE

Стиль DRAW\_LINE рисует заданным цветом линию по значениям индикаторного буфера. Толщину, цвет и стиль отображения линии можно задавать как [директивами компилятора](#), так и динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет создавать "живые" индикаторы, которые меняют свой вид в зависимости от текущей ситуации.

Количество требуемых буферов для построения DRAW\_LINE – 1.

Пример индикатора, рисующего линию ценам закрытия баров Close. Цвет, толщина и стиль линии меняются случайным образом каждые N=5 тиков.



Обратите внимание, первоначально для графического построения `plot1` со стилем DRAW\_LINE свойства задаются с помощью директивы компилятора [#property](#), а затем в функции [OnCalculate\(\)](#) эти три свойства задаются случайным образом. Параметр N вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```

//+-----+
//|                                              DRAW_LINE.mq5  |
//|                                              Copyright 2011, MetaQuotes Software Corp.  |
//|                                              https://www.mql5.com  |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_LINE"
#property description "Рисует линию заданным цветом по ценам Close"
#property description "Цвет, толщина и стиль линии меняется случайным образом"

```

```

#property description "через каждые N тиков"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- свойства линии заданы с помощью директив компилятора
#property indicator_label1 "Line" // название построения для "Окна данных"
#property indicator_type1 DRAW_LINE // тип графического построения - линия
#property indicator_color1 clrRed // цвет линии
#property indicator_style1 STYLE_SOLID // стиль линии
#property indicator_width1 1 // толщина линии

//--- input параметр
input int      N=5; // кол-во тиков для изменения

//--- индикаторный буфер для построения
double          LineBuffer[];

//--- массив для хранения цветов
color colors[]={clrRed,clrBlue,clrGreen};

//--- массив для хранения стилей отрисовки линии
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT2};

//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
    //--- связывание массива и индикаторного буфера
    SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);

    //--- инициализация генератора псевдослучайных чисел
    MathStrand(GetTickCount());

    //---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=0;

    //--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;

    //--- если накопилось критическое число тиков
}

```

```

if(ticks>=N)
{
    //--- меняем свойства линии
    ChangeLineAppearance();
    //--- сбрасываем счетчик тиков в ноль
    ticks=0;
}

//--- блок расчета значений индикатора
for(int i=0;i<rates_total;i++)
{
    LineBuffer[i]=close[i];
}

//--- вернем значение prev_calculated для следующего вызова функции
return(rates_total);
}

//+-----+
//| Изменяет внешний вид отображаемой линии в индикаторе |
//+-----+
void ChangeLineAppearance()
{
    //--- строка для формирования информации о свойствах линии
    string comm="";
    //--- блок изменения цвета линии
    //--- получим случайное число
    int number=MathRand();
    //--- делитель числа равен размеру массива colors[]
    int size=ArraySize(colors[]);
    //--- получим индекс для выбора нового цвета как остаток от целочисленного деления
    int color_index=number%size;
    //--- установим цвет как свойство PLOT_LINE_COLOR
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
    //--- запишем цвет линии
    comm=comm+(string)colors[color_index];

    //--- блок изменения толщины линии
    number=MathRand();
    //--- получим толщину как остаток от целочисленного деления
    int width=number%5; // толщина задается от 0 до 4
    //--- установим цвет как свойство PLOT_LINE_WIDTH
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
    //--- запишем толщину линии
    comm=comm+", Width="+IntegerToString(width);

    //--- блок изменения стиля линии
    number=MathRand();
    //--- делитель числа равен размеру массива styles
    size=ArraySize(styles);
}

```

```
//--- получим индекс для выбора нового стиля как остаток от целочисленного деления
int style_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- запишем стиль линии
comm=EnumToString(styles[style_index])+", "+comm;
//--- выведем информацию на график через комментарий
Comment(comm);
}
```

## DRAW\_SECTION

Стиль DRAW\_SECTION рисует заданным цветом отрезки по значениям индикаторного буфера. Толщину, цвет и стиль отображения линии можно задавать так же, как и для стиля [DRAW\\_LINE](#) - [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет "оживить" индикаторы, чтобы они меняли свой вид в зависимости от текущей ситуации.

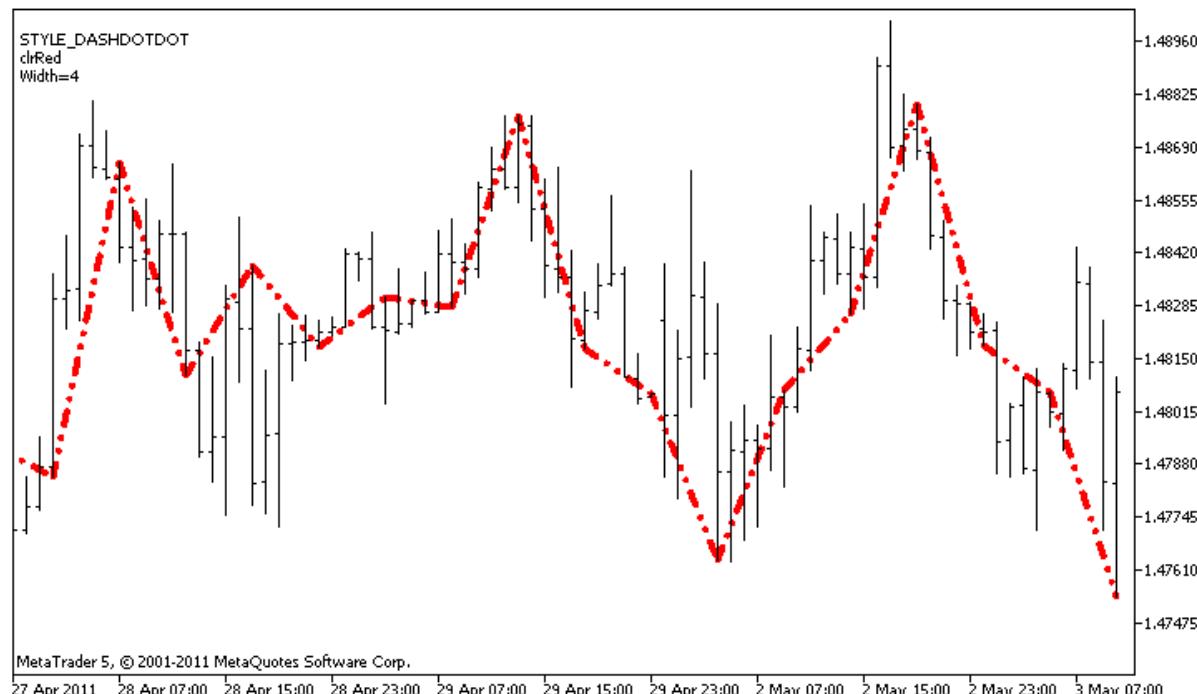
Секции рисуются от одного непустого значения до другого непустого значения индикаторного буфера, пустые значения пропускаются. Чтобы указать, какое значение следует считать "пустым", установите это значение в свойстве [PLOT\\_EMPTY\\_VALUE](#). Например, если индикатор должен рисоваться отрезками по ненулевым значениям, то нужно задать нулевое значение в качестве пустого:

```
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
PlotIndexSetDouble(индекс_построения_DRAW_SECTION, PLOT_EMPTY_VALUE, 0);
```

Всегда явно заполняйте значениями все элементы индикаторного буфера, неотрисовываемым элементам задавайте пустое значение.

Количество требуемых буферов для построения DRAW\_SECTION – 1.

Пример индикатора, рисующего отрезки между ценами High и Low. Цвет, толщина и стиль всех секций меняются случайным образом каждые N тиков.



Обратите внимание, первоначально для графического построения `plot1` со стилем DRAW\_SECTION свойства задаются с помощью директивы компилятора [#property](#), а затем в функции [OnCalculate\(\)](#) эти три свойства задаются случайным образом. Параметр N вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```
//+-----+
```

```

//| DRAW_SECTION.mq5 |
//| Copyright 2011, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_SECTION"
#property description "Рисует прямыми отрезками через каждые bars баров"
#property description "Цвет, толщина и стиль секций меняется случайным образом"
#property description "через каждые N тиков"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots    1
//--- plot Section
#property indicator_label1  "Section"
#property indicator_type1   DRAW_SECTION
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input параметр
input int      bars=5;           // длина секций в барах
input int      N=5;              // кол-во тиков для изменения стиля секций
//--- индикаторный буфер для построения
double        SectionBuffer[];
//--- вспомогательная переменная для вычисления концов секций
int          divider;
//--- массив для хранения цветов
color colors[]={clrRed,clrBlue,clrGreen};
//--- массив для хранения стилей отрисовки линии
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT};
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- связывание массива и индикаторного буфера
SetIndexBuffer(0,SectionBuffer,INDICATOR_DATA);
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- проверим параметр индикатора
if(bars<=0)
{
PrintFormat("Недопустимое значение параметра bar=%d",bars);
return(INIT_PARAMETERS_INCORRECT);
}
else divider=2*bars;
}

```

```

//---+
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=0;
//--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;
//--- если накопилось критическое число тиков
    if(ticks>=N)
    {
        //--- меняем свойства линии
        ChangeLineAppearance();
        //--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }

//--- номер бара, с которого начнем расчет значений индикатора
    int start=0;
//--- если индикатор уже рассчитывали раньше, то установим start на предыдущий бар
    if(prev_calculated>0) start=prev_calculated-1;
//--- здесь все расчеты значений индикатора
    for(int i=start;i<rates_total;i++)
    {
        //--- получим остаток от деления номера бара на 2*bars
        int rest=i%divider;
        //--- если номер бара делится без остатка на 2*bars
        if(rest==0)
        {
            //--- конец отрезка установим на цену High этого бара
            SectionBuffer[i]=high[i];
        }
        //--- если остаток от деления равен bars,
        else
        {
            //--- конец отрезка установим на цену Low этого бара
            if(rest==bars) SectionBuffer[i]=low[i];
        }
    }
}

```

```

        //--- если ничего не подошло, то этот бар пропускаем - ставим значение 0
        else SectionBuffer[i]=0;
    }
}

//--- вернем значение prev_calculated для следующего вызова функции
return(rates_total);
}

//+-----+
//| Изменяет внешний вид секций в индикаторе |
//+-----+

void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах линии
string comm="";
//--- блок изменения цвета линии
int number=MathRand(); // получим случайное число
//--- делитель числа равен размеру массива colors[]
int size=ArraySize(colors);
//--- получим индекс для выбора нового цвета как остаток от целочисленного деления
int color_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- запишем цвет линии
comm=comm+"\r\n"+(string)colors[color_index];

//--- блок изменения толщины линии
number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5; // толщина задается от 0 до 4
//--- установим толщину
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем толщину линии
comm=comm+"\r\nWidth="+IntegerToString(width);

//--- блок изменения стиля линии
number=MathRand();
//--- делитель числа равен размеру массива styles
size=ArraySize(styles);
//--- получим индекс для выбора нового стиля как остаток от целочисленного деления
int style_index=number%size;
//--- установим стиль линий
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- запишем стиль линии
comm="\r\n"+EnumToString(styles[style_index])+" "+comm;
//--- выведем информацию на график через комментарий
Comment(comm);
}

```

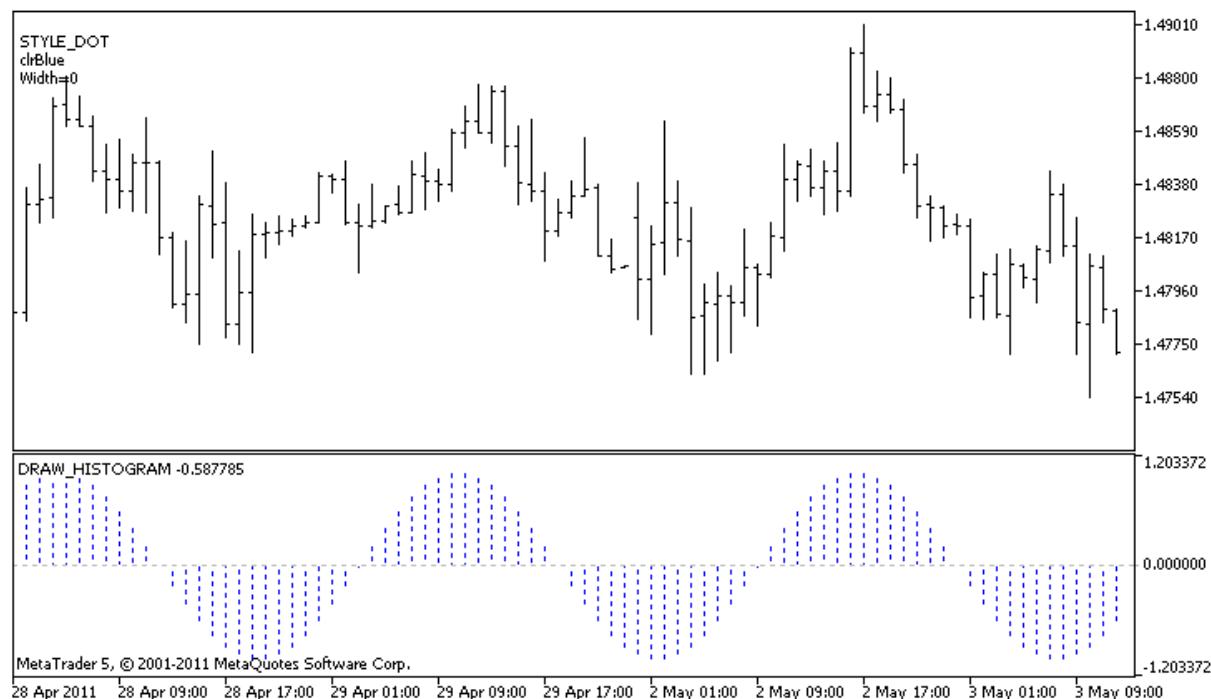
## DRAW\_HISTOGRAM

Стиль DRAW\_HISTOGRAM рисует заданным цветом гистограмму столбиками от нуля до указанного значения. Значения берутся из индикаторного буфера. Толщину, цвет и стиль отображения стобика можно задавать так же, как и для стиля [DRAW\\_LINE](#) - [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет изменять вид гистограммы в зависимости от текущей ситуации.

Так как на каждом баре рисуется столбик от нулевого уровня, то DRAW\_HISTOGRAM лучше использовать для отображения в отдельном подокне графика. Чаще всего этот тип графического построения используется для создания индикаторов осцилляторного типа, например, [Bears Power](#) или [OsMA](#). Для пустых неотображаемых значений достаточно указывать нулевое значение.

Количество требуемых буферов для построения DRAW\_HISTOGRAM – 1.

Пример индикатора, рисующего заданным цветом синусоиду по функции [MathSin\(\)](#). Цвет, толщина и стиль **всех** столбиков гистограммы меняются случайным образом каждые N тиков. Параметр bars задает период синусоиды, то есть через заданное количество баров синусоида будет повторяться по циклу.



Обратите внимание, что первоначально для графического построения `plot1` со стилем DRAW\_HISTOGRAM свойства задаются с помощью директивы компилятора [#property](#), а затем в функции [OnCalculate\(\)](#) эти три свойства задаются случайным образом. Параметр N вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```
//+
//|
//|          DRAW_HISTOGRAM.mq5  |
//| Copyright 2011, MetaQuotes Software Corp.  |
//| https://www.mql5.com  |
```

```

//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_HISTOGRAM"
#property description "Рисует синусоиду гистограммой в отдельном окне"
#property description "Цвет и толщина столбиков меняется случайным образом"
#property description "через каждые N тиков"
#property description "Параметр bars задает количество баров в цикле синусоиды"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots    1
//--- plot Histogram
#property indicator_label1  "Histogram"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input параметры
input int      bars=30;           // период синусоиды в барах
input int      N=5;              // кол-во тиков для изменения гистограммы
//--- indicator buffers
double        HistogramBuffer[];
//--- множитель для получения угла 2Pi в радианах при умножении на параметр bars
double        multiplier;
//--- массив для хранения цветов
color colors[]={clrRed,clrBlue,clrGreen};
//--- массив для хранения стилей отрисовки линии
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT2};
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- вычислим множитель
if(bars>1)multiplier=2.*M_PI/bars;
else
{
PrintFormat("Задайте значение bars=%d больше 1",bars);
//--- досрочное прекращение работы индикатора
return(INIT_PARAMETERS_INCORRECT);
}
//---
return(INIT_SUCCEEDED);
}

```

```

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=0;
//--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;
//--- если накопилось критическое число тиков
    if(ticks>=N)
    {
        //--- меняем свойства линии
        ChangeLineAppearance();
        //--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }

//--- вычисления значений индикатора
    int start=0;
//--- если расчет уже производился на предыдущем запуске OnCalculate
    if(prev_calculated>0) start=prev_calculated-1; // установим начало расчетов с предыдущего запуска
//--- заполняем индикаторный буфер значениями
    for(int i=start;i<rates_total;i++)
    {
        HistogramBuffer[i]=sin(i*multiplier);
    }
//--- вернем значение prev_calculated для следующего вызова функции
    return(rates_total);
}

//+-----+
//| Изменяет внешний вид линий в индикаторе
//+-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах линии
    string comm="";
//--- блок изменения цвета линии
    int number=MathRand(); // получим случайное число
//--- делитель числа равен размеру массива colors[]
    int size=ArraySize(colors);
}

```

```
//--- получим индекс для выбора нового цвета как остаток от целочисленного деления
int color_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- запишем цвет линии
comm=comm+"\r\n"+(string)colors[color_index];

//--- блок изменения толщины линии
number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5; // толщина задается от 0 до 4
//--- установим толщину
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем толщину линии
comm=comm+"\r\nWidth="+IntegerToString(width);

//--- блок изменения стиля линии
number=MathRand();
//--- делитель числа равен размеру массива styles
size=ArraySize(styles);
//--- получим индекс для выбора нового стиля как остаток от целочисленного деления
int style_index=number%size;
//--- установим стиль линий
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- запишем стиль линии
comm="\r\n"+EnumToString(styles[style_index])+" "+comm;
//--- выведем информацию на график через комментарий
Comment(comm);
}
```

## DRAW\_HISTOGRAM2

Стиль DRAW\_HISTOGRAM2 рисует заданным цветом гистограмму - вертикальные отрезки по значениям двух индикаторных буферов. Толщину, цвет и стиль отображения отрезков можно задавать так же, как и для стиля [DRAW\\_LINE](#) - [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет изменять вид гистограммы в зависимости от текущей ситуации.

Стиль DRAW\_HISTOGRAM можно использовать как в отдельном подокне графика, так и в главном окне. Для пустых значений отрисовка не производится, все значения в индикаторных буферах нужно устанавливать явным образом. Инициализация буферов пустым значением не производится.

Количество требуемых буферов для построения DRAW\_HISTOGRAM2 – 2.

Пример индикатора, рисующего на каждом баре вертикальный отрезок заданного цвета и толщины между ценами Open и Close. Цвет, толщина и стиль **всех** столбиков гистограммы меняются случайным образом каждые N тиков. При запуске индикатора в функции [OnInit\(\)](#) случайным образом устанавливается номер дня недели, для которого гистограмма не будет рисоваться - [invisible\\_day](#). Для этого задается пустое значение [PLOT\\_EMPTY\\_VALUE=0](#):

```
//--- установим пустое значение
PlotIndexSetDouble(индекс_построения_DRAW_SECTION,PLOT_EMPTY_VALUE,0);
```



Обратите внимание, что первоначально для графического построения `plot1` со стилем DRAW\_HISTOGRAM2 свойства задаются с помощью директивы компилятора `#property`, а затем в функции [OnCalculate\(\)](#) эти три свойства задаются случайным образом. Параметр N вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```
//+-----+
//|                               DRAW_HISTOGRAM2.mq5 |
//| Copyright 2011, MetaQuotes Software Corp. |
```

```
//+-----+
//| Copyright 2011, MetaQuotes Software Corp.
//| https://www.mql5.com
//| Version 1.00
//| DRAW_HISTOGRAM2
//| Рисует на каждом баре отрезок между Open и Close
//| Цвет, толщина и стиль меняется случайным образом
//| через каждые N тиков

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plot Histogram_2
#property indicator_label1 "Histogram_2"
#property indicator_type1 DRAW_HISTOGRAM2
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameters
input int      N=5;           // кол-во тиков для изменения гистограммы
//--- indicator buffers
double         Histogram_2Buffer1[];
double         Histogram_2Buffer2[];
//--- день недели, для которого индикатор не рисуется
int   invisible_day;
//--- массив для хранения цветов
color colors[]={clrRed,clrBlue,clrGreen};
//--- массив для хранения стилей отрисовки линии
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT2};
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,Histogram_2Buffer1,INDICATOR_DATA);
    SetIndexBuffer(1,Histogram_2Buffer2,INDICATOR_DATA);
//--- установим пустое значение
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- получим случайное число от 0 до 5
    invisible_day=MathRand()%6;
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
```

```

int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
    //--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;
    //--- если накопилось критическое число тиков
    if(ticks>=N)
    {
        //--- меняем свойства линии
        ChangeLineAppearance();
        //--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }

    //--- вычисления значений индикатора
    int start=0;
    //--- для получения дня недели по времени открытия каждого бара
    MqlDateTime dt;
    //--- если расчет уже производился на предыдущем запуске OnCalculate
    if(prev_calculated>0) start=prev_calculated-1; // установим начало расчетов с предыдущего
    //--- заполняем индикаторный буфер значениями
    for(int i=start;i<rates_total;i++)
    {
        TimeToStruct(time[i],dt);
        if(dt.day_of_week==invisible_day)
        {
            Histogram_2Buffer1[i]=0;
            Histogram_2Buffer2[i]=0;
        }
        else
        {
            Histogram_2Buffer1[i]=open[i];
            Histogram_2Buffer2[i]=close[i];
        }
    }
    //--- вернем значение prev_calculated для следующего вызова функции
    return(rates_total);
}
//+-----+
// | Изменяет внешний вид линий в индикаторе |

```

```

//+-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах линии
string comm="";
//--- блок изменения цвета линии
int number=MathRand(); // получим случайное число
//--- делитель числа равен размеру массива colors[]
int size=ArraySize(colors);
//--- получим индекс для выбора нового цвета как остаток от целочисленного деления
int color_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- запишем цвет линии
comm=comm+"\r\n"+(string)colors[color_index];

//--- блок изменения толщины линии
number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5; // толщина задается от 0 до 4
//--- установим толщину линий
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем толщину линии
comm=comm+"\r\nWidth="+IntegerToString(width);

//--- блок изменения стиля линии
number=MathRand();
//--- делитель числа равен размеру массива styles
size=ArraySize(styles);
//--- получим индекс для выбора нового стиля как остаток от целочисленного деления
int style_index=number%size;
//--- установим стиль линий
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- запишем стиль линии
comm="\r\n"+EnumToString(styles[style_index])+" "+comm;
//--- добавим информацию о дне, который пропускается в расчетах
comm="\r\nНеотрисовываемый день - "+EnumToString((ENUM_DAY_OF_WEEK)invisible_day)+comm;
//--- выведем информацию на график через комментарий
Comment(comm);
}

```

## DRAW\_ARROW

Стиль DRAW\_ARROW рисует на графике заданным цветом стрелки (символы из набора [Wingdings](#)) по значению индикаторного буфера. Толщину и цвет символов можно задавать так же, как и для стиля [DRAW\\_LINE](#) - [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет изменять вид индикатора в зависимости от текущей ситуации.

Код символа для вывода на график задается с помощью свойства [PLOT\\_ARROW](#).

```
//--- зададим код символа из шрифта Wingdings для отрисовки в PLOT_ARROW
PlotIndexSetInteger(0, PLOT_ARROW, code);
```

По умолчанию значение PLOT\_ARROW=159 (кружок).

Каждая стрелка фактически представляет собой символ, который имеет высоту и точку привязки, и может закрывать собою некоторую важную информацию на графике (например, цену закрытия на баре). Поэтому можно дополнительно указать вертикальное смещение в пикселях, которое не зависит от масштаба графика. На это указанное количество пикселей стрелки будут визуально смещены по вертикали, хотя сами значения индикатора останутся те же самые:

```
//--- зададим смещение стрелок по вертикали в пикселях
PlotIndexSetInteger(0, PLOT_ARROW_SHIFT, shift);
```

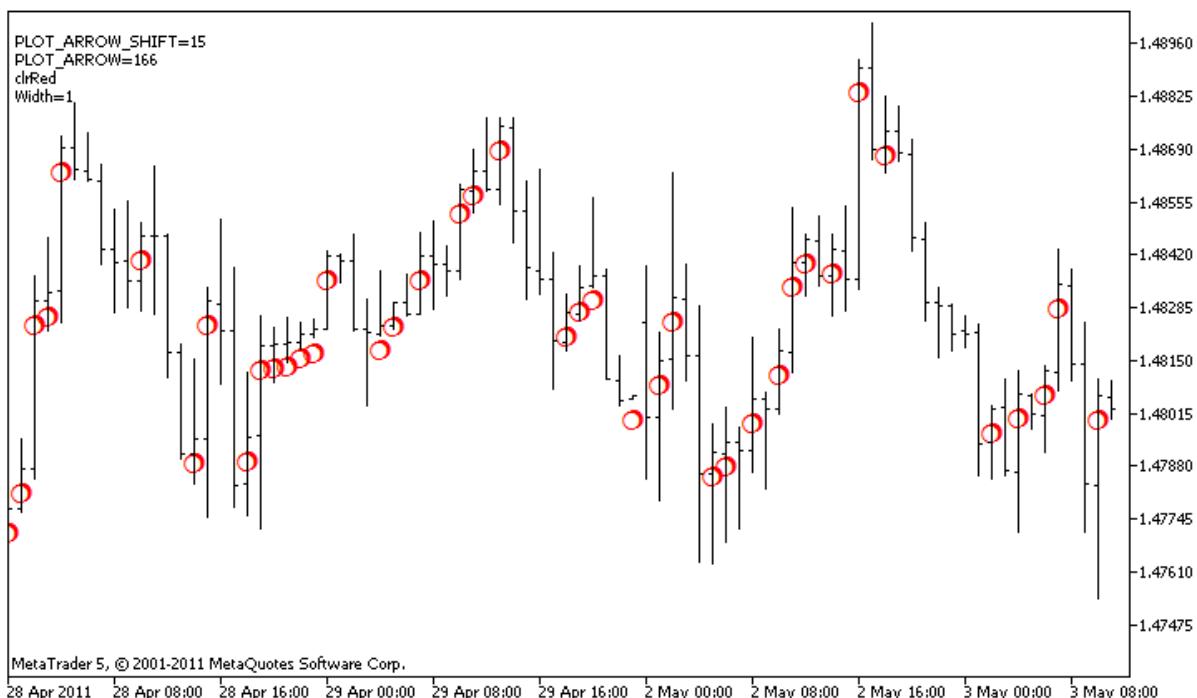
Отрицательное значение PLOT\_ARROW\_SHIFT означает смещение стрелок вверх, положительное значение смещает стрелки вниз.

Стиль DRAW\_ARROW можно использовать как в отдельном подокне графика, так и в главном окне. Пустые значения не отрисовываются и не отображаются в "Окне данных", все значения в индикаторных буферах нужно устанавливать явным образом. Инициализация буферов пустым значением не производится.

```
//--- установим пустое значение
PlotIndexSetDouble(индекс_построения_DRAW_ARROW, PLOT_EMPTY_VALUE, 0);
```

Количество требуемых буферов для построения DRAW\_ARROW – 1.

Пример индикатора, рисующего стрелки на каждом баре, у которого цена закрытия Close больше цены закрытия предыдущего бара. Цвет, толщина, смещение и код символа **всех** стрелок меняются случайным образом каждые N тиков.



В примере первоначально для графического построения `plot1` со стилем `DRAW_ARROW` свойства, цвет и размер задаются с помощью директивы компилятора `#property`, а затем в функции `OnCalculate()` свойства задаются случайным образом. Параметр `N` вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```

//+-----+
//|                               DRAW_ARROW.mq5 |
//|           Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_ARROW"
#property description "Рисует на графике стрелки, задаваемые символами Unicode"
#property description "Цвет, размер, смещение и код символа стрелки меняется случайным"
#property description "через каждые N тиков"
#property description "Параметр code задает базовое значение: код=159 (кружок)"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots    1
//--- plot Arrows
#property indicator_label1  "Arrows"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrGreen
#property indicator_width1  1
//--- input параметры

```

```

input int      N=5;           // кол-во тиков для изменения
input ushort   code=159;      // код символа для отрисовки в DRAW_ARROW
//--- индикаторный буфер для построения
double        ArrowsBuffer[];
//--- массив для хранения цветов
color colors[]={clrRed,clrBlue,clrGreen};
//+-----+
//| Custom indicator initialization function          |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,ArrowsBuffer,INDICATOR_DATA);
//--- зададим код символа для отрисовки в PLOT_ARROW
    PlotIndexSetInteger(0,PLOT_ARROW,code);
//--- зададим смещение стрелок по вертикали в пикселях
    PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,5);
//--- установим в качестве пустого значения 0
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
//--- return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function               |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=0;
//--- считаем тики для изменения цвета, размера, смещения и кода стрелки
    ticks++;
//--- если накопилось критическое число тиков
    if(ticks>=N)
    {
//--- меняем свойства линии
        ChangeLineAppearance();
//--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }
}

//--- блок расчета значений индикатора

```

```

int start=1;
if(prev_calculated>0) start=prev_calculated-1;
//--- цикл расчета
for(int i=1;i<rates_total;i++)
{
    //--- если текущая цена Close больше предыдущей, ставим стрелку
    if(close[i]>close[i-1])
        ArrowsBuffer[i]=close[i];
    //--- в противном случае указываем нулевое значение
    else
        ArrowsBuffer[i]=0;
}
//--- return value of prev_calculated for next call
return(rates_total);
}

//+-----+
//| Изменяет внешний вид символов в индикаторе |
//+-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах индикатора
string comm="";
//--- блок изменения цвета стрелки
int number=MathRand(); // получим случайное число
//--- делитель числа равен размеру массива colors[]
int size=ArraySize(colors[]);
//--- получим индекс для выбора нового цвета как остаток от целочисленного деления
int color_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- запишем цвет линии
comm=comm+"\r\n"+(string)colors[color_index];

//--- блок изменения размера стрелок
number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5; // размер задается от 0 до 4
//--- установим цвет как свойство PLOT_LINE_WIDTH
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем размер стрелок
comm=comm+"\r\nWidth="+IntegerToString(width);

//--- блок изменения кода стрелки (PLOT_ARROW)
number=MathRand();
//--- получим остаток от целочисленного деления для вычисления нового кода стрелки (от
int code_add=number%20;
//--- установим новый код символа как сумму code+code_add
PlotIndexSetInteger(0,PLOT_ARROW,code+code_add);
//--- запишем код символа PLOT_ARROW
}

```

```
comm="\r\n"+PLOT_ARROW=IntegerToString(code+code_add)+comm;

//--- блок изменения смещения стрелок по вертикали в пикселях
number=MathRand();
//--- получим смещение как остаток от целочисленного деления
int shift=20-number%41;
//--- установим новое смещение от -20 до 20
PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
//--- запишем смещение PLOT_ARROW_SHIFT
comm="\r\n"+PLOT_ARROW_SHIFT=IntegerToString(shift)+comm;

//--- выведем информацию на график через комментарий
Comment(comm);
}
```

## DRAW\_ZIGZAG

Стиль DRAW\_ZIGZAG рисует заданным цветом отрезки по значениям двух индикаторных буферов. Этот стиль очень похож на [DRAW\\_SECTION](#), но в отличие от последнего позволяет рисовать вертикальные отрезки в пределах одного бара, если для этого бара заданы значения для обоих индикаторных буферов. Отрезки рисуются от значения в первом буфере до значения во втором индикаторном буфере. Ни один из буферов не может содержать только пустые значения, так как в этом случае отрисовка не происходит.

Толщину, цвет и стиль отображения линии можно задавать так же, как и для стиля [DRAW\\_SECTION](#) - [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет "оживить" индикаторы, чтобы они меняли свой вид в зависимости от текущей ситуации.

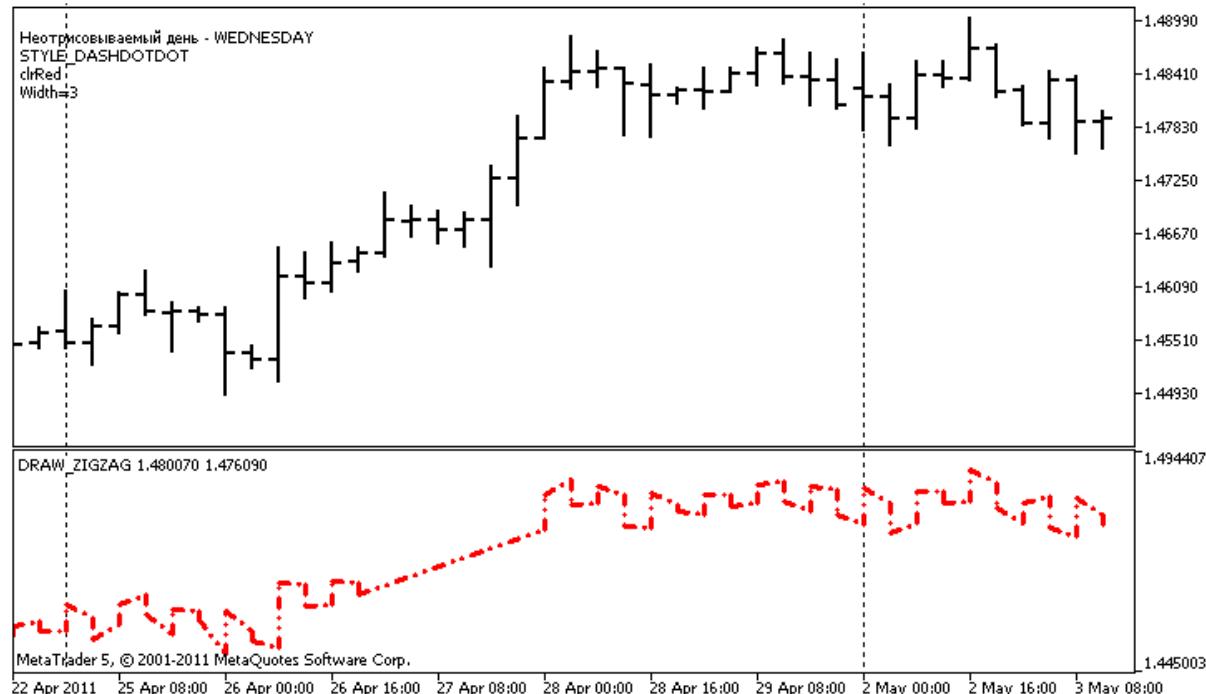
Секции рисуются от одного непустого значения одного буфера до непустого значения другого индикаторного буфера. Чтобы указать, какое значение следует считать "пустым", установите это значение в свойстве [PLOT\\_EMPTY\\_VALUE](#):

```
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
PlotIndexSetDouble(индекс_построения_DRAW_ZIGZAG, PLOT_EMPTY_VALUE, 0);
```

Всегда явно заполняйте значениями индикаторные буфера, на пропускаемых барах указывайте в буфере пустое значение.

Количество требуемых буферов для построения DRAW\_ZIGZAG – 2.

Пример индикатора, рисующего пилю по ценам High и Low. Цвет, толщина и стиль линий зигзага меняются случайным образом каждые N тиков.



Обратите внимание, первоначально для графического построения `plot1` со стилем DRAW\_ZIGZAG свойства задаются с помощью директивы компилятора [#property](#), а затем в функции [OnCalculate\(\)](#) эти три свойства задаются случайным образом. Параметр N вынесен во [внешние параметры](#)

индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```

//+-----+
//|                                         DRAW_ZIGZAG.mq5 |
//|                                         Copyright 2011, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_ZIGZAG"
#property description "Рисует прямыми отрезками \"пилу\", пропуская бары одного дня"
#property description "День пропусков выбирается случайным образом при запуске индикатора"
#property description "Цвет, толщина и стиль отрезков меняется случайным образом"
#property description "образом через каждые N тиков"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plot ZigZag
#property indicator_label1 "ZigZag"
#property indicator_type1 DRAW_ZIGZAG
#property indicator_color1 clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input параметры
input int      N=5;                      // кол-во тиков для изменения
//--- indicator buffers
double        ZigZagBuffer1[];
double        ZigZagBuffer2[];
//--- день недели, для которого индикатор не рисуется
int          invisible_day;
//--- массив для хранения цветов
color         colors[]={clrRed,clrBlue,clrGreen};
//--- массив для хранения стилей отрисовки линии
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT2};
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- связывание массивов и индикаторных буферов
    SetIndexBuffer(0,ZigZagBuffer1,INDICATOR_DATA);
    SetIndexBuffer(1,ZigZagBuffer2,INDICATOR_DATA);
//--- получим случайное число от 0 до 6, для этого дня индикатор не рисуется
    invisible_day=MathRand()%6;
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
}

```

```

    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
    PlotIndexSetString(0,PLOT_LABEL,"ZigZag1;ZigZag2");
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=0;
//--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;
//--- если накопилось достаточное число тиков
    if(ticks>=N)
    {
        //--- меняем свойства линии
        ChangeLineAppearance();
        //--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }

//--- структура времени понадобится для получения дня недели каждого бара
MqlDateTime dt;

//--- позиция начала расчетов
int start=0;
//--- если индикатор рассчитывался на предыдущем тике, то начинаем расчет с предпоследнего
if(prev_calculated!=0) start=prev_calculated-1;
//--- цикл расчетов
for(int i=start;i<rates_total;i++)
{
    //--- запишем время открытия бара в структуру
    TimeToStruct(time[i],dt);
    //--- если день недели этого бара равен invisible_day
    if(dt.day_of_week==invisible_day)
    {
        //--- запишем пустые значения в буферы для этого бара
        ZigZagBuffer1[i]=0;
    }
}

```

```

        ZigZagBuffer2[i]=0;
    }
//--- если день недели подходящий, то заполняем буферы
else
{
//--- если номер бара четный
if(i%2==0)
{
//--- пишем в 1-ый буфер High, во 2-ой Low
ZigZagBuffer1[i]=high[i];
ZigZagBuffer2[i]=low[i];
}
//--- номер бара нечетный
else
{
//--- заполняем бар в обратном порядке
ZigZagBuffer1[i]=low[i];
ZigZagBuffer2[i]=high[i];
}
}
}

//--- return value of prev_calculated for next call
return(rates_total);
}

//-----+
//| Изменяет внешний вид отрезков в зигзаге |
//-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах ZigZag
string comm="";
//--- блок изменения цвета зигзага
int number=MathRand(); // получим случайное число
//--- делитель числа равен размеру массива colors[]
int size=ArraySize(colors);
//--- получим индекс для выбора нового цвета как остаток от целочисленного деления
int color_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- запишем цвет линии
comm=comm+"\r\n"+(string)colors[color_index];

//--- блок изменения толщины линии
number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5; // толщина задается от 0 до 4
//--- установим цвет как свойство PLOT_LINE_WIDTH
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем толщину линии
}

```

```
comm=comm+"\r\nWidth="+IntegerToString(width);  
  
//--- блок изменения стиля линии  
number=MathRand();  
//--- делитель числа равен размеру массива styles  
size=ArraySize(styles);  
//--- получим индекс для выбора нового стиля как остаток от целочисленного деления  
int style_index=number%size;  
//--- установим цвет как свойство PLOT_LINE_COLOR  
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);  
//--- запишем стиль линии  
comm="\r\n"+EnumToString(styles[style_index])+"  
";  
//--- добавим информацию о дне, который пропускается в расчетах  
comm="\r\nНеотрисовываемый день - "+EnumToString((ENUM_DAY_OF_WEEK)invisible_day)+  
//--- выведем информацию на график через комментарий  
Comment(comm);  
}
```

## DRAW\_FILLING

Стиль DRAW\_FILLING рисует цветную область между значениями двух индикаторных буферов. Фактически этот стиль рисует две линии и закрашивает пространство между ними одним из двух заданных цветов. Предназначен для создания индикаторов, рисующих каналы. Ни один из буферов не может содержать только пустые значения, так как в этом случае отрисовка не происходит.

Можно задавать два цвета заливки:

- первый цвет для тех областей, где значения в первом индикаторном буфере больше значений во втором индикаторном буфере;
- второй цвет для тех областей, где значения во втором индикаторном буфере больше значений в первом индикаторном буфере.

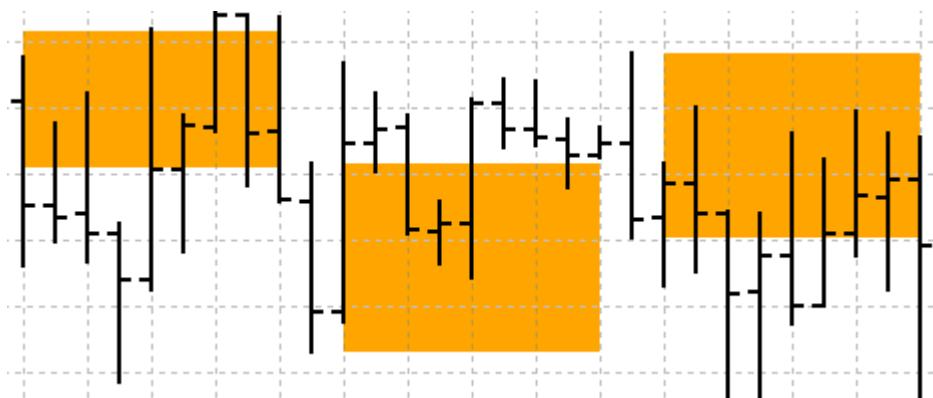
Цвет заливки можно задавать [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет "оживить" индикаторы, чтобы они меняли свой вид в зависимости от текущей ситуации.

Индикатор рассчитывается для всех баров, для которых значения обоих индикаторных буферов не равны 0 и не равны пустому значению. Чтобы указать, какое значение следует считать "пустым", установите это значение в свойстве [PLOT\\_EMPTY\\_VALUE](#):

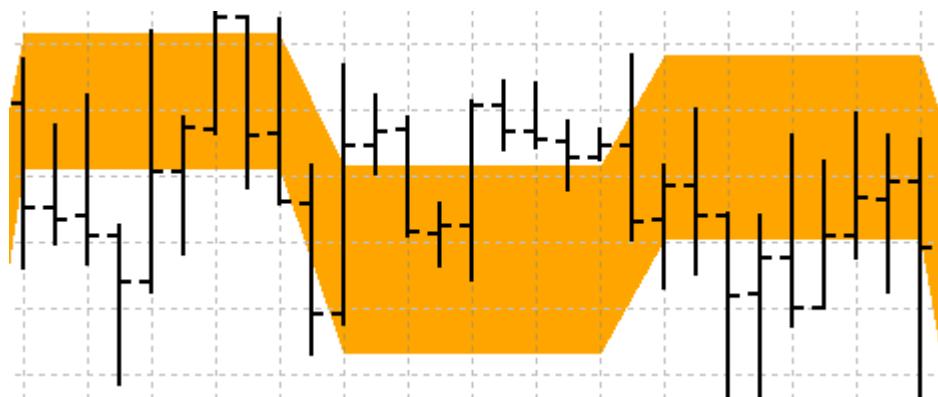
```
#define INDICATOR_EMPTY_VALUE -1.0
...
//--- значение INDICATOR_EMPTY_VALUE (пустое значение) не будет участвовать в расчете
PlotIndexSetDouble(индекс_построения_DRAW_FILLING, PLOT_EMPTY_VALUE, INDICATOR_EMPTY_VALUE)
```

Отрисовка на барах, которые не участвуют в расчете индикатора, будет зависеть от значений в индикаторных буферах:

- Бары, для которых значения обоих индикаторных буферов равны 0, не участвуют в отрисовке индикатора. То есть область с нулевыми значениями не будет закрашиваться.



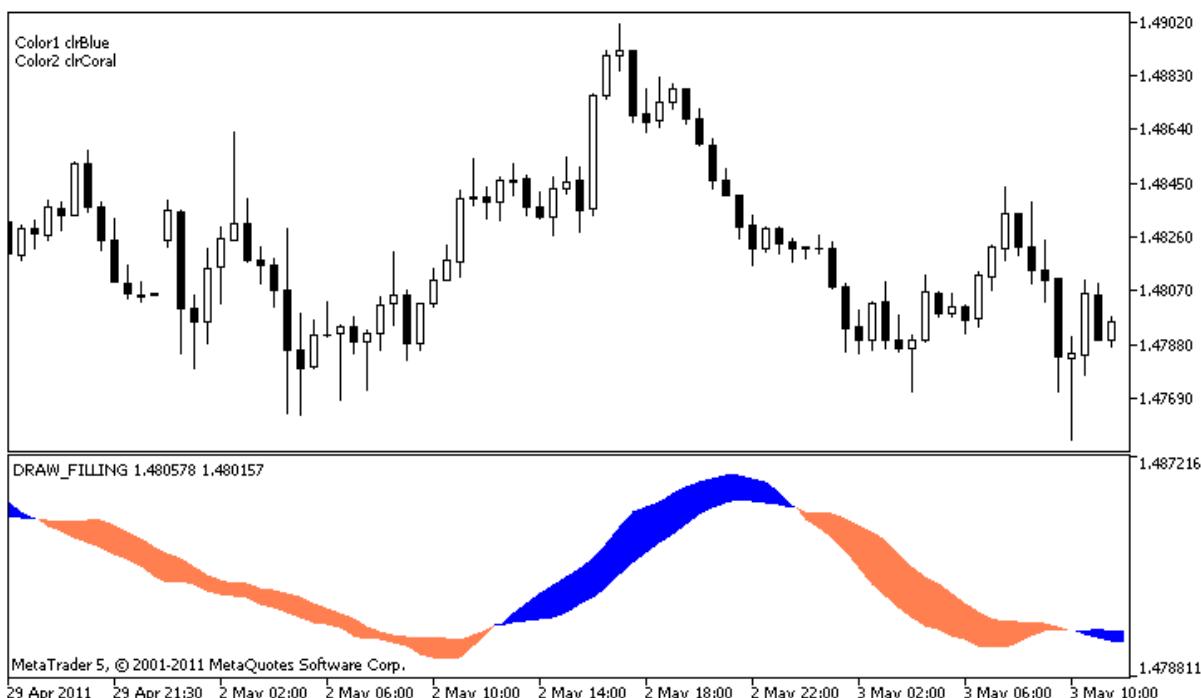
- Бары, для которых значения индикаторных буферов равны "пустому значению", участвуют в отрисовке индикатора. Область с пустыми значениями будет закрашиваться таким образом, чтобы соединять области со значащими значениями.



При этом важно отметить, что если "пустое значение" равняется нулю, то бары, не участвующие в расчете индикатора, также будут закрашиваться.

Количество требуемых буферов для построения DRAW\_FILLING – 2.

Пример индикатора, рисующего в отдельном окне канал между двумя скользящими средними с разными периодами усреднения. Изменение цвета при пересечении средних визуально показывает смену восходящей и нисходящей тенденций. Цвета меняются случайным образом каждые N тиков. Параметр N вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).



Обратите внимание, первоначально для графического построения `plot1` со стилем `DRAW_FILLING` два цвета задаются с помощью директивы компилятора [`#property`](#), а затем в функции [`OnCalculate\(\)`](#) новые цвета задаются случайным образом.

```
//+-----+
//|                               DRAW_FILLING.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
```

```

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_FILLING"
#property description "Рисует в отдельном окне канал между двумя средними"
#property description "Цвет заливки канала меняется случайным образом"
#property description "через каждые N тиков"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plot Intersection
#property indicator_label1 "Intersection"
#property indicator_type1 DRAW_FILLING
#property indicator_color1 clrRed,clrBlue
#property indicator_width1 1
//--- input параметры
input int    Fast=13;           // период быстрой скользящей средней
input int    Slow=21;           // период медленной скользящей средней
input int    shift=1;           // сдвиг средних в будущее (положительный)
input int    N=5;               // кол-во тиков для изменения
//--- индикаторные буфера
double       IntersectionBuffer1[];
double       IntersectionBuffer2[];
int         fast_handle;
int         slow_handle;
//--- массив для хранения цветов
color colors[]={clrRed,clrBlue,clrGreen,clrAquamarine,clrBlanchedAlmond,clrBrown,clrC
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,IntersectionBuffer1,INDICATOR_DATA);
    SetIndexBuffer(1,IntersectionBuffer2,INDICATOR_DATA);
//---
    PlotIndexSetInteger(0,PLOT_SHIFT,shift);
//---
    fast_handle=iMA(_Symbol,_Period,Fast,0,MODE_SMA,PRICE_CLOSE);
    slow_handle=iMA(_Symbol,_Period,Slow,0,MODE_SMA,PRICE_CLOSE);
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,

```

```

        const int prev_calculated,
        const datetime &time[],
        const double &open[],
        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
        static int ticks=0;
//--- считаем тики для изменения стиля, цвета и толщины линии
        ticks++;
//--- если накопилось достаточное число тиков
        if(ticks>=N)
        {
//--- меняем свойства линии
            ChangeLineAppearance();
//--- сбрасываем счетчик тиков в ноль
            ticks=0;
        }

//--- делаем первый расчет индикатора или данные изменились и требуется полный перерасчет
        if(prev_calculated==0)
        {
//--- копируем все значения индикаторов в соответствующие буферы
            int copied1=CopyBuffer(fast_handle,0,0,rates_total,IntersectionBuffer1);
            int copied2=CopyBuffer(slow_handle,0,0,rates_total,IntersectionBuffer2);
        }
        else // экономно заполняем только те данные, которые обновились
        {
//--- получим разницу в барах между текущим и предыдущим запуском OnCalculate()
            int to_copy=rates_total-prev_calculated;
//--- если разницы нет, то все равно будем копировать одно значение - на нулевом баре
            if(to_copy==0) to_copy=1;
//--- копируем to_copy значений в самый конец индикаторных буферов
            int copied1=CopyBuffer(fast_handle,0,0,to_copy,IntersectionBuffer1);
            int copied2=CopyBuffer(slow_handle,0,0,to_copy,IntersectionBuffer2);
        }
//--- return value of prev_calculated for next call
        return(rates_total);
    }
//-----+
//| Изменяет цвета заливки канала |
//-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах линии
    string comm="";
}

```

```
//--- блок изменения цвета линии
int number=MathRand(); // получим случайное число
//--- делитель числа равен размеру массива colors[]
int size=ArraySize(colors);

//--- получим индекс для выбора нового цвета как остаток от целочисленного деления
int color_index1=number%size;
//--- установим первый цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,colors[color_index1]);
//--- запишем первый цвет
comm=comm+"\r\nColor1 "+(string)colors[color_index1];

//--- получим индекс для выбора нового цвета как остаток от целочисленного деления
number=MathRand(); // получим случайное число
int color_index2=number%size;
//--- установим второй цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,colors[color_index2]);
//--- запишем второй цвет
comm=comm+"\r\nColor2 "+(string)colors[color_index2];
//--- выведем информацию на график через комментарий
Comment(comm);
}
```

## DRAW\_BARS

Стиль DRAW\_BARS рисует бары по значениям четырех индикаторных буферов, в которых содержатся цены Open, High, Low и Close. Предназначен для создания собственных индикаторов в виде баров, в том числе в отдельном подокне графика и по другим финансовым инструментам.

Цвет баров можно задавать [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет "оживить" индикаторы, чтобы они меняли свой вид в зависимости от текущей ситуации.

Индикатор рисуется только для тех баров, для которых заданы непустые значения **всех** четырех индикаторных буферов. Чтобы указать, какое значение следует считать "пустым", установите это значение в свойстве [PLOT\\_EMPTY\\_VALUE](#):

```
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
PlotIndexSetDouble(индекс_построения_DRAW_BARS, PLOT_EMPTY_VALUE, 0);
```

Всегда явно заполняйте значениями индикаторные буфера, на пропускаемых барах указывайте в буфере пустое значение.

Количество требуемых буферов для построения DRAW\_BARS – 4. Все буфера для построения должны идти последовательно один за другим в заданном порядке: Open, High, Low и Close. Ни один из буферов не может содержать только пустые значения, так как в этом случае отрисовка не происходит.

Пример индикатора, рисующего в отдельном окне бары по указанному финансовому инструменту. Цвет баров меняется случайным образом каждые N тиков. Параметр N вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).



Обратите внимание, первоначально для графического построения **plot1** со стилем **DRAW\_BARS** цвет задается с помощью директивы компилятора **#property**, а затем в функции **OnCalculate()** цвет выбирается случайным образом из заранее подготовленного списка.

```
//+-----+
//|                                         DRAW_BARS.mq5  |
//|                                         Copyright 2011, MetaQuotes Software Corp.  |
//|                                         https://www.mql5.com  |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_BARS"
#property description "Рисует в отдельном окне бары по выбранному символу"
#property description "Цвет и толщина баров, а также символ, меняются случайным "
#property description "образом через каждые N тиков"

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots 1
//--- plot Bars
#property indicator_label1 "Bars"
#property indicator_type1 DRAW_BARS
#property indicator_color1 clrGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input параметры
input int      N=5;                      // количество тиков для смены вида
input int      bars=500;                   // сколько баров показывать
input bool     messages=false;             // вывод сообщений в лог "Эксперты"
//--- индикаторные буфера
double        BarsBuffer1[];
double        BarsBuffer2[];
double        BarsBuffer3[];
double        BarsBuffer4[];
//--- имя символа
string       symbol;
//--- массив для хранения цветов
color         colors[]={clrRed,clrBlue,clrGreen,clrPurple,clrBrown,clrIndianRed};
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- если bars слишком мало - досрочно завершаем работу
if(bars<50)
{
Comment("Укажите большее количество баров! Работа индикатора прекращена");
}
}
```

```

        return (INIT_PARAMETERS_INCORRECT);
    }

//--- indicator buffers mapping
SetIndexBuffer(0,BarsBuffer1,INDICATOR_DATA);
SetIndexBuffer(1,BarsBuffer2,INDICATOR_DATA);
SetIndexBuffer(2,BarsBuffer3,INDICATOR_DATA);
SetIndexBuffer(3,BarsBuffer4,INDICATOR_DATA);

//--- имя символа, по которому рисуются бары
symbol=_Symbol;

//--- установим отображение символа
PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symbol+" Close");
IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_BARS("+symbol+")");

//--- пустое значение
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);

//---

return (INIT_SUCCEEDED);
}

//-----+
//| Custom indicator iteration function
//-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=0;

//--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;

//--- если накопилось достаточное число тиков
    if(ticks>=N)
    {
        //--- выберем новый символ из окна "Обзор рынка"
        symbol=GetRandomSymbolName();

        //--- меняем свойства линии
        ChangeLineAppearance();

        int tries=0;
        //--- сделаем 5 попыток заполнить буферы ценами из symbol
        while(!CopyFromSymbolToBuffers(symbol,rates_total) && tries<5)
        {
            //--- счетчик вызовов функции CopyFromSymbolToBuffers()
            tries++;
        }
    }
}

```

```

//--- сбрасываем счетчик тиков в ноль
ticks=0;
}

//--- return value of prev_calculated for next call
return(rates_total);
}

//+-----+
//| Заполняет индикаторные буферы ценами |
//+-----+

bool CopyFromSymbolToBuffers(string name,int total)
{
//--- в массив rates[] будем копировать цены Open, High, Low и Close
MqlRates rates[];
//--- счетчик попыток
int attempts=0;
//--- сколько скопировано
int copied=0;
//--- делаем 25 попыток получить таймсерию по нужному символу
while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
{
    Sleep(100);
    attempts++;
    if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attempts);
}
//--- если не удалось скопировать достаточное количество баров
if(copied!=bars)
{
    //--- сформируем строку сообщения
    string comm=StringFormat("Для символа %s удалось получить только %d баров из %d",
                                name,
                                copied,
                                bars
                            );
    //--- выведем сообщение в комментарий на главное окно графика
    Comment(comm);
    //--- выводим сообщения
    if(messages) Print(comm);
    return(false);
}
else
{
    //--- установим отображение символа
    PlotIndexSetString(0,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;"+name+"");
    IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_BARS("+name+ ")");
}
//--- инициализируем буфера пустыми значениями
ArrayInitialize(BarsBuffer1,0.0);
ArrayInitialize(BarsBuffer2,0.0);
ArrayInitialize(BarsBuffer3,0.0);

```

```

        ArrayInitialize(BarsBuffer4,0.0);
//--- копируем цены в буферы
for(int i=0;i<copied;i++)
{
    //--- вычислим соответствующий индекс для буферов
    int buffer_index=total-copied+i;
    //--- записываем цены в буферы
    BarsBuffer1[buffer_index]=rates[i].open;
    BarsBuffer2[buffer_index]=rates[i].high;
    BarsBuffer3[buffer_index]=rates[i].low;
    BarsBuffer4[buffer_index]=rates[i].close;
}
return(true);
}

//+-----+
//| Возвращает случайным образом символ из Market Watch           |
//+-----+
string GetRandomSymbolName()
{
//--- количество символов, показываемых в окне "Обзор рынка"
int symbols=SymbolsTotal(true);
//--- позиция символа в списке - случайное число от 0 до symbols
int number=MathRand()%symbols;
//--- вернем имя символа по указанной позиции
return SymbolName(number,true);
}

//+-----+
//| Изменяет внешний вид баров                                       |
//+-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах баров
string comm="";
//--- блок изменения цвета баров
int number=MathRand(); // получим случайное число
//--- делитель числа равен размеру массива colors[]
int size=ArraySize(colors[]);
//--- получим индекс для выбора нового цвета как остаток от целочисленного деления
int color_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- запишем цвет линии
comm=comm+"\r\n"+(string)colors[color_index];

//--- блок изменения толщины баров
number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5; // толщина задается от 0 до 4
//--- установим цвет как свойство PLOT_LINE_WIDTH
}

```

```
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем толщину линии
comm=comm+"\r\nWidth="+IntegerToString(width);

//--- запишем имя символа
comm="\r\n"+symbol+comm;

//--- выведем информацию на график через комментарий
Comment(comm);
}
```

## DRAW\_CANDLES

Стиль DRAW\_CANDLES рисует японские свечи по значениям четырёх индикаторных буферов, в которых содержатся цены Open, High, Low и Close. Предназначен для создания собственных индикаторов в виде свечей, в том числе в отдельном подокне графика и по другим финансовым инструментам.

Цвет свечей можно задавать [директивами компилятора](#) или динамически с помощью функции `PlotIndexSetInteger()`. Динамическое изменение свойств графического построения позволяет "оживить" индикаторы, чтобы они меняли свой вид в зависимости от текущей ситуации.

Индикатор рисуется только для тех баров, для которых заданы непустые значения **всех** четырёх индикаторных буферов. Чтобы указать, какое значение следует считать "пустым", установите это значение в свойстве `PLOT_EMPTY_VALUE`:

```
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
PlotIndexSetDouble(индекс_построения_DRAW_CANDLES, PLOT_EMPTY_VALUE, 0);
```

Всегда явно заполняйте значениями индикаторные буфера, на пропускаемых барах указывайте в буфере пустое значение.

Количество требуемых буферов для построения DRAW\_CANDLES – 4. Все буфера для построения должны идти последовательно один за другим в заданном порядке: Open, High, Low и Close. Ни один из буферов не может содержать только пустые значения, так как в этом случае отрисовка не происходит.

Для стиля DRAW\_CANDLES можно задавать от одного до трёх цветов, в зависимости от этого меняется внешний вид свечей. Если указан только один цвет, то все свечи на графике будут полностью окрашены этим цветом.

```
//--- одинаковые свечи, окрашенные в один цвет
#property indicator_label1 "One color candles"
#property indicator_type1 DRAW_CANDLES
//--- указан только один цвет, поэтому все свечи будут одного цвета
#property indicator_color1 clrGreen
```

Если указать два цвета через запятую, то контуры свечи будут отрисовываться первым цветом, а тело вторым цветом.

```
//--- цвет свечей отличается от цвета теней
#property indicator_label1 "Two color candles"
#property indicator_type1 DRAW_CANDLES
//--- тени и контур свечей зеленого цвета, тело белого цвета
#property indicator_color1 clrGreen,clrWhite
```

Для того чтобы по разному показывать растущие и падающие свечи, необходимо указать через запятую 3 цвета. В этом случае контур свечи будет нарисован первым цветом, а цвет бычьей и медвежьей свечей будет задаваться вторым и третьим цветом.

```
//--- цвет свечей отличается от цвета теней
#property indicator_label1 "One color candles"
#property indicator_type1 DRAW_CANDLES
//--- тени и контур зелёного цвета, тело бычьей свечи белого цвета, тело медвежьей све
```

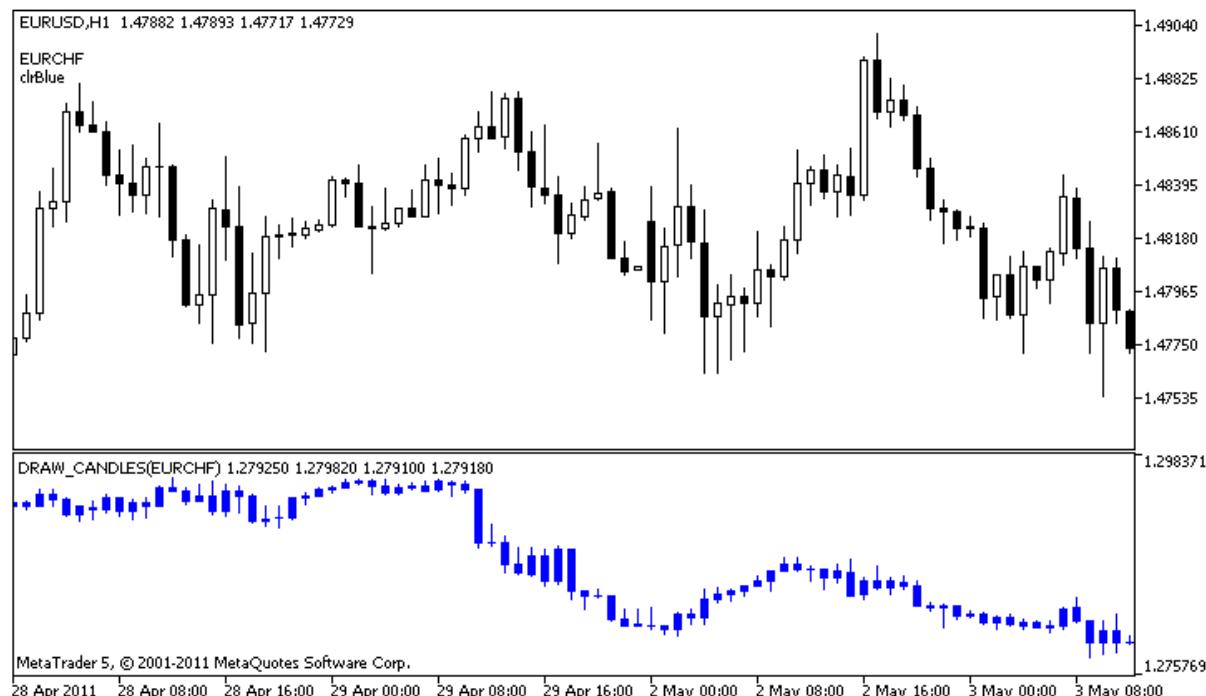
```
#property indicator_color1 clrGreen,clrWhite,clrRed
```

Таким образом, с помощью стиля DRAW\_CANDLES можно создавать собственные пользовательские варианты раскраски свечей. Все цвета можно также менять динамически в процессе работы индикатора с помощью функции PlotIndexSetInteger(индекс\_построения\_DRAW\_CANDLES, PLOT\_LINE\_COLOR, номер\_модификатора, цвет) , где номер\_модификатора может иметь следующие значения:

- 0 - цвет контура и теней
- 1- цвет тела бычьей свечи
- 2 - цвет тела медвежьей свечи

```
//--- установим цвет контура и теней
PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrBlue);
//--- установим цвет тела для бычьей свечи
PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrGreen);
//--- установим цвет тела для медвежьей свечи
PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrRed);
```

Пример индикатора, рисующего в отдельном окне японские свечи по указанному финансовому инструменту. Цвет свечей меняется случайным образом каждые N тиков. Параметр N вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).



Обратите внимание, первоначально для графического построения `plot1` цвет задается с помощью директивы компилятора [#property](#), а затем в функции [OnCalculate\(\)](#) выбирается новый цвет случайным образом из заранее подготовленного списка.

```
//+-----+
//|
//|          DRAW_CANDLES.mq5  |
//|          Copyright 2011, MetaQuotes Software Corp.  |
//|          https://www.mql5.com  |
```

```

//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_CANDLES."
#property description "Рисует в отдельном окне разным цветом свечи по случайнно выбранн"
#property description " "
#property description "Цвет и толщина свечей, а также символ, меняются"
#property description "случайным образом через каждые N тиков."

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots    1
//--- plot Bars
#property indicator_label1  "DRAW_CANDLES1"
#property indicator_type1   DRAW_CANDLES
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- input параметры
input int      N=5;           // количество тиков для смены вида
input int      bars=500;        // сколько баров показывать
input bool     messages=false;  // вывод сообщений в лог "Эксперты"
//--- индикаторные буферы
double        Candle1Buffer1[];
double        Candle1Buffer2[];
double        Candle1Buffer3[];
double        Candle1Buffer4[];
//--- имя символа
string       symbol;
//--- массив для хранения цветов
color colors[]={clrRed,clrBlue,clrGreen,clrPurple,clrBrown,clrIndianRed};
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- если bars слишком мало - досрочно завершаем работу
if(bars<50)
{
Comment("Укажите большее количество баров! Работа индикатора прекращена");
return(INIT_PARAMETERS_INCORRECT);
}
//--- indicator buffers mapping
SetIndexBuffer(0,Candle1Buffer1,INDICATOR_DATA);
SetIndexBuffer(1,Candle1Buffer2,INDICATOR_DATA);
SetIndexBuffer(2,Candle1Buffer3,INDICATOR_DATA);
}

```

```

        SetIndexBuffer(3,Candle1Buffer4,INDICATOR_DATA);
//--- пустое значение
        PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- имя символа, по которому рисуются бары
        symbol=_Symbol;
//--- установим отображение символа
        PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symbol+" Close");
        IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_CANDLES("+symbol+ ")");
//---
        return(INIT_SUCCEEDED);
    }
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=INT_MAX-100;
//--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;
//--- если накопилось достаточное число тиков
    if(ticks>=N)
    {
//--- выберем новый символ из окна "Обзор рынка"
        symbol=GetRandomSymbolName();
//--- сменим вид
        ChangeLineAppearance();
//--- выберем новый символ из окна "Обзор рынка"
        int tries=0;
//--- сделаем 5 попыток заполнить буфера plot1 ценами из symbol
        while(!CopyFromSymbolToBuffers(symbol,rates_total,0,
                                       Candle1Buffer1,Candle1Buffer2,Candle1Buffer3,Candle1Buffer4)
              && tries<5)
        {
//--- счетчик вызовов функции CopyFromSymbolToBuffers()
            tries++;
        }
//--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }
//--- return value of prev_calculated for next call

```

```

        return(rates_total);
    }

//+-----+
//| Заполняет указанную свечу |
//+-----+

bool CopyFromSymbolToBuffers(string name,
                             int total,
                             int plot_index,
                             double &buff1[],
                             double &buff2[],
                             double &buff3[],
                             double &buff4[])
{
    //--- в массив rates[] будем копировать цены Open, High, Low и Close
    MqlRates rates[];
    //--- счетчик попыток
    int attempts=0;
    //--- сколько скопировано
    int copied=0;
    //--- делаем 25 попыток получить таймсерию по нужному символу
    while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
    {
        Sleep(100);
        attempts++;
        if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attempts);
    }
    //--- если не удалось скопировать достаточное количество баров
    if(copied!=bars)
    {
        //--- сформируем строку сообщения
        string comm=StringFormat("Для символа %s удалось получить только %d баров из %d",
                                 name,
                                 copied,
                                 bars
                                );
        //--- выведем сообщение в комментарий на главное окно графика
        Comment(comm);
        //--- выводим сообщения
        if(messages) Print(comm);
        return(false);
    }
    else
    {
        //--- установим отображение символа
        PlotIndexSetString(plot_index,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;
    }
    //--- инициализируем буфера пустыми значениями
    ArrayInitialize(buff1,0.0);
}

```

```

        ArrayInitialize(buff2,0.0);
        ArrayInitialize(buff3,0.0);
        ArrayInitialize(buff4,0.0);

//--- на каждом тике копируем цены в буферы
    for(int i=0;i<copied;i++)
    {
        //--- вычислим соответствующий индекс для буферов
        int buffer_index=total-copied+i;
        //--- записываем цены в буферы
        buff1[buffer_index]=rates[i].open;
        buff2[buffer_index]=rates[i].high;
        buff3[buffer_index]=rates[i].low;
        buff4[buffer_index]=rates[i].close;
    }
    return(true);
}

//+-----+
//| Возвращает случайным образом символ из Market Watch |
//+-----+

string GetRandomSymbolName()
{
//--- количество символов, показываемых в окне "Обзор рынка"
    int symbols=SymbolsTotal(true);
//--- позиция символа в списке - случайное число от 0 до symbols
    int number=MathRand()%symbols;
//--- вернем имя символа по указанной позиции
    return SymbolName(number,true);
}

//+-----+
//| Изменяет внешний вид баров |
//+-----+

void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах баров
    string comm="";
//--- блок изменения цвета баров
    int number=MathRand(); // получим случайное число
//--- делитель числа равен размеру массива colors[]
    int size=ArraySize(colors);
//--- получим индекс для выбора нового цвета как остаток от целочисленного деления
    int color_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- запишем цвет
    comm=comm+"\r\n"+(string)colors[color_index];
//--- запишем имя символа
    comm="\r\n"+symbol+comm;
//--- выведем информацию на график через комментарий
    Comment(comm);
}

```

{}

## DRAW\_COLOR\_LINE

Стиль DRAW\_COLOR\_LINE является цветным вариантом стиля [DRAW\\_LINE](#), он также рисует линию по значениям индикаторного буфера. Но у этого стиля, как у всех цветных стилей, имеющих в название COLOR, есть дополнительный специальный индикаторный буфер, который хранит индекс (номер) цвета из специального заданного массива цветов. Таким образом, цвет каждого участка линии можно задать, если указать индекса цвета, которым линия должна рисоваться на данном баре.

Толщину, стиль и цвета для отображения линии можно задавать как [директивами компилятора](#), так и динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет создавать "живые" индикаторы, которые меняют свой вид в зависимости от текущей ситуации.

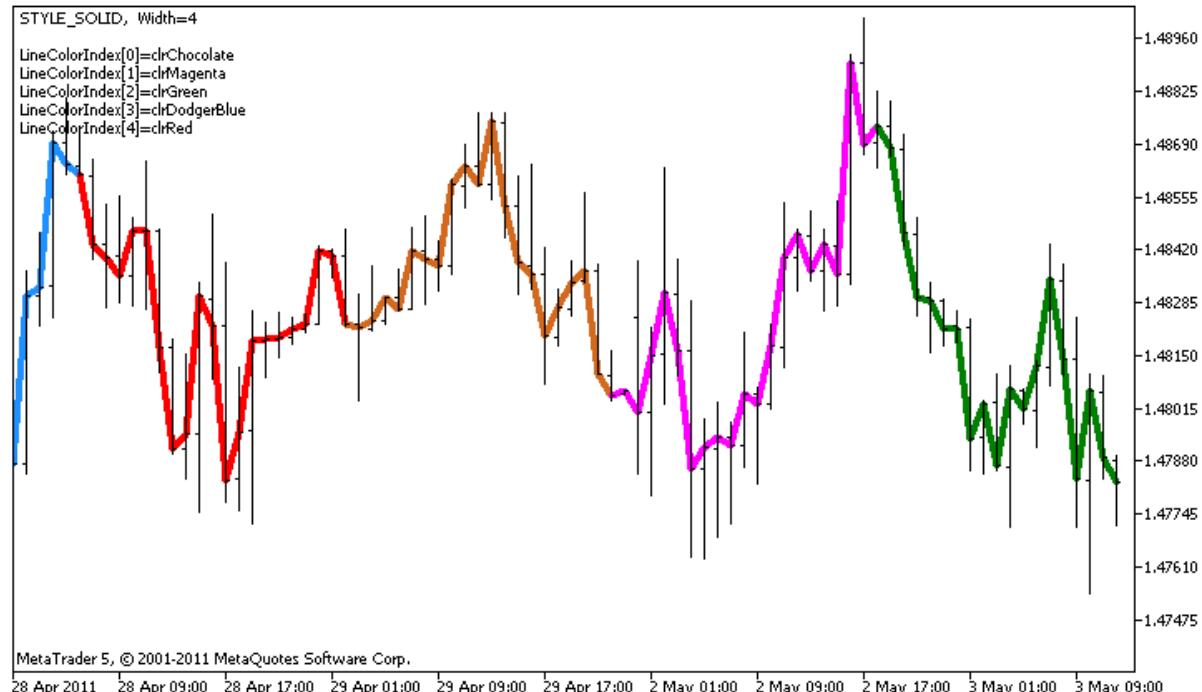
Количество требуемых буферов для построения DRAW\_COLOR\_LINE – 2:

- один буфер для хранения значений индикатора, по которым рисуется линия;
- один буфер для хранения индекса цвета, которым рисуется линия на каждом баре.

Цвета можно задавать директивой компилятора `#property indicator_color1` через запятую. Количество цветов не может превышать 64.

```
//--- зададим 5 цветов для раскраски каждого бара (они хранятся в специальном массиве)
#property indicator_color1 clrRed,clrBlue,clrGreen,clrOrange,clrDeepPink // (можно у
```

Пример индикатора, рисующего линию по ценам закрытия баров Close. Толщина и стиль линии меняются случайным образом каждые N=5 тиков.



Цвета, которыми рисуются участки линии, также меняются случайным образом в пользовательской функции `ChangeColors()`.

```
//+-----+
```

```
//| Изменяет цвет участков линии |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- количество цветов
int size=ArraySize(cols);
//---
string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- для каждого цветового индекса зададим новый цвет случайным образом
for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
{
//--- получим случайное число
int number=MathRand();
//--- получим индекс в массиве col[] как остаток от целочисленного деления
int i=number%size;
//--- установим цвет для каждого индекса как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0, // номер графического стиля
PLOT_LINE_COLOR, // идентификатор свойства
plot_color_ind, // индекс цвета, куда запишем цвет
cols[i]); // новый цвет
//--- запишем цвета
comm=comm+StringFormat("LineColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString(
ChartSetString(0,CHART_COMMENT,comm));
}
//---
}
```

В примере показана особенность "цветных" версий индикаторов - для смены цвета участка линии не требуется менять значения в буфере ColorLineColors[] (который содержит индексы цветов). Достаточно задать новые цвета в специальном массиве. Это позволяет быстро поменять цвет сразу для всего графического построения, изменения только небольшой массив цветов с помощью функции [PlotIndexSetInteger\(\)](#).

Обратите внимание, первоначально для графического построения **plot1** со стилем DRAW\_COLOR\_LINE свойства задаются с помощью директивы компилятора [#property](#), а затем в функции [OnCalculate\(\)](#) эти три свойства задаются случайным образом.

Параметры **N** и **Length** (длина цветных участков в барах) вынесены во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```
//+
//| DRAW_COLOR_LINE.mq5 |
//| Copyright 2011, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```

#property description "Индикатор для демонстрации DRAW_COLOR_LINE"
#property description "Рисует цветными кусками по 20 баров линию по ценам Close"
#property description "Толщина, стиль и цвет участков линии меняется случайным"
#property description "образом через каждые N тиков"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 1
//--- plot ColorLine
#property indicator_label1 "ColorLine"
#property indicator_type1 DRAW_COLOR_LINE
//--- зададим 5 цветов для раскраски каждого бара (они хранятся в специальном массиве)
#property indicator_color1 clrRed,clrBlue,clrGreen,clrOrange,clrDeepPink // (можно указать
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input параметры
input int      N=5;           // кол-во тиков для изменения
input int      Length=20;       // длина каждого участка цвета в барах
int          line_colors=5;    // количество заданных цветов равно 5 - смотри выше #property
//--- буфер для отрисовки
double        ColorLineBuffer[];
//--- буфер для хранения цвета отрисовки линии на каждом баре
double        ColorLineColors[];

//--- массив для хранения цветов содержит 7 элементов
color colors[]={clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGold};
//--- массив для хранения стилей отрисовки линии
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDASH,
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- связывание массива и индикаторного буфера
    SetIndexBuffer(0,ColorLineBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,ColorLineColors,INDICATOR_COLOR_INDEX);
//--- инициализация генератора псевдослучайных чисел
    MathStrand(GetTickCount());
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
```

```

        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
        static int ticks=0;
//--- считаем тики для изменения стиля, цвета и толщины линии
        ticks++;
//--- если накопилось критическое число тиков
        if(ticks>=N)
        {
//--- меняем свойства линии
            ChangeLineAppearance();
//--- меняем цвета, которыми рисуются цветные участки линии
            ChangeColors(colors,5);
//--- сбрасываем счетчик тиков в ноль
            ticks=0;
        }

//--- блок расчета значений индикатора
        for(int i=0;i<rates_total;i++)
        {
//--- запишем значение индикатора в буфер
            ColorLineBuffer[i]=close[i];
//--- теперь случайным образом зададим для этого бара индекс цвета
            int color_index=i%(5*Length);
            color_index=color_index/Length;
//--- для этого бара линия будет рисоваться цветом, который хранится под номером
            ColorLineColors[i]=color_index;
        }

//--- вернем значение prev_calculated для следующего вызова функции
        return(rates_total);
    }
//-----+
//| Изменяет цвет участков линии |
//-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- количество цветов
    int size=ArraySize(cols);
//---
    string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";
//--- для каждого цветового индекса зададим новый цвет случайным образом
    for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
    {
}

```

```

//--- получим случайное число
int number=MathRand();

//--- получим индекс в массиве col[] как остаток от целочисленного деления
int i=number%size;

//--- установим цвет для каждого индекса как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0, // номер графического стиля
                     PLOT_LINE_COLOR, // идентификатор свойства
                     plot_color_ind, // индекс цвета, куда запишем цвет
                     cols[i]); // новый цвет

//--- запишем цвета
comm=comm+StringFormat("LineColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString(plot_color));
ChartSetString(0,CHART_COMMENT,comm);
}

//---

}

//+-----+
//| Изменяет внешний вид отображаемой линии в индикаторе |
//+-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах линии
string comm="";
//--- блок изменения толщины линии
int number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5; // толщина задается от 0 до 4
//--- установим цвет как свойство PLOT_LINE_WIDTH
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем толщину линии
comm=comm+" Width="+IntegerToString(width);

//--- блок изменения стиля линии
number=MathRand();
//--- делитель числа равен размеру массива styles
int size=ArraySize(styles);
//--- получим индекс для выбора нового стиля как остаток от целочисленного деления
int style_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- запишем стиль линии
comm=EnumToString(styles[style_index])+", "+comm;
//--- выведем информацию на график через комментарий
Comment(comm);
}

```

## DRAW\_COLOR\_SECTION

Стиль DRAW\_COLOR\_SECTION является цветным вариантом стиля [DRAW\\_SECTION](#), но в отличие от последнего, позволяет рисовать каждую секцию собственным цветом. Стиль DRAW\_COLOR\_SECTION, как и все цветные стили, имеющие в название COLOR, содержит дополнительный специальный индикаторный буфер для хранения индекса (номера) цвета из специального заданного массива цветов. Таким образом, цвет каждой секции можно задать, если указать индекса цвета для бара, на который приходится конец секции.

Толщину, цвет и стиль отрезков можно задавать так же, как и для стиля [DRAW\\_SECTION](#) - [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет "оживить" индикаторы, чтобы они меняли свой вид в зависимости от текущей ситуации.

Секции рисуются от одного непустого значения до другого непустого значения индикаторного буфера, пустые значения пропускаются. Чтобы указать, какое значение следует считать "пустым", установите это значение в свойстве [PLOT\\_EMPTY\\_VALUE](#). Например, если индикатор должен рисоваться отрезками по ненулевым значениям, то нужно задать нулевое значение в качестве пустого:

```
//--- значение 0 (пустое значение) не будет участвовать в отрисовке  
PlotIndexSetDouble(индекс_построения_DRAW_COLOR_SECTION,PLOT_EMPTY_VALUE,0);
```

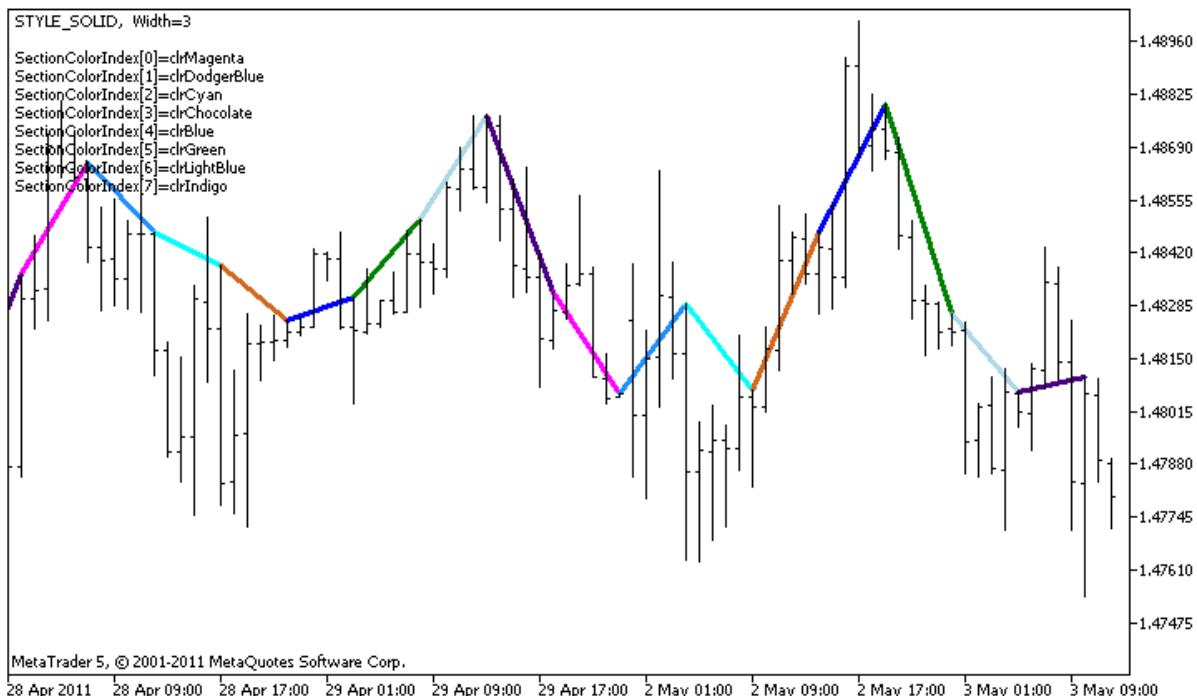
Всегда явно заполняйте значениями все элементы индикаторного буфера, неотрисовываемым элементам задавайте пустое значение.

Количество требуемых буферов для построения DRAW\_COLOR\_SECTION – 2:

- один буфер для хранения значений индикатора, по которым рисуется линия;
- один буфер для хранения индекса цвета, которым рисуется секция (имеет смысл задавать только для непустых значений).

Цвета можно задавать директивой компилятора `#property indicator_color1` через запятую. Количество цветов не может превышать 64.

Пример индикатора, рисующего разноцветные секции длиной 5 баров по ценам High. Цвет, толщина и стиль секций меняются случайным образом каждые N тиков.



Обратите внимание, первоначально для графического построения `plot1` со стилем `DRAW_COLOR_SECTION` свойства задается 8 цветов с помощью директивы компилятора `#property`. Затем в функции `OnCalculate()` цвета задаются случайным образом из массива цветов `colors[]`.

Параметр `N` вынесен во внешние параметры индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```

//+-----+
//|                               DRAW_COLOR_SECTION.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_COLOR_SECTION"
#property description "Рисует цветными отрезками длиной в заданное число баров"
#property description "Цвет, толщина и стиль секций меняется случайным образом"
#property description "через каждые N тиков"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots    1
//--- plot ColorSection
#property indicator_label1  "ColorSection"
#property indicator_type1   DRAW_COLOR_SECTION
//--- зададим 8 цветов для раскраски секций (они хранятся в специальном массиве)
#property indicator_color1  clrRed,clrGold,clrMediumBlue,clrLime,clrMagenta,clrBrown,clrOrange,clrGrey
#property indicator_style1   STYLE_SOLID

```

```

#property indicator_width1 1
//--- input параметры
input int      N=5;                                // кол-во тиков для изменения
input int      bars_in_section=5;                  // длина секций в барах
//--- вспомогательная переменная для вычисления концов секций
int           divider;
int           color_sections;
//--- буфер для отрисовки
double        ColorSectionBuffer[];
//--- буфер для хранения цвета отрисовки линии на каждом баре
double        ColorSectionColors[];
//--- массив для хранения цветов содержит 14 элементов
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurp
};
//--- массив для хранения стилей отрисовки линии
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTD
//+-----+
//| Custom indicator initialization function          |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,ColorSectionBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,ColorSectionColors,INDICATOR_COLOR_INDEX);
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- количество цветов для раскраски секций
    color_sections=8;    // см. комментарий к свойству #property indicator_color1
//--- проверим параметр индикатора
    if(bars_in_section<=0)
    {
        PrintFormat("Недопустимое значение длины секции=%d",bars_in_section);
        return(INIT_PARAMETERS_INCORRECT);
    }
    else divider=color_sections*bars_in_section;
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function            |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
```

```

        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
{
    static int ticks=0;
//--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;
//--- если накопилось критическое число тиков
    if(ticks>=N)
    {
        //--- меняем свойства линии
        ChangeLineAppearance();
        //--- меняем цвета, которыми рисуются секции
        ChangeColors(colors,color_sections);
        //--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }

//--- номер бара, с которого начнем расчет значений индикатора
    int start=0;
//--- если индикатор уже рассчитывали раньше, то установим start на предыдущий бар
    if(prev_calculated>0) start=prev_calculated-1;
//--- здесь все расчеты значений индикатора
    for(int i=start;i<rates_total;i++)
    {
        //--- если номер бара делится без остатка на длину_секции, значит это конец секции
        if(i%bars_in_section==0)
        {
            //--- конец отрезка установим на цену High этого бара
            ColorSectionBuffer[i]=high[i];
            //--- остаток от деления номера бара на длина_секции*количество_цветов
            int rest=i%divider;
            //получим номер цвета = от 0 до количество_цветов-1
            int color_index=rest/bars_in_section;
            ColorSectionColors[i]=color_index;
        }
        //--- если остаток от деления равен bars,
        else
        {
            //--- если ничего не подошло, то этот бар пропускаем - ставим значение 0
            ColorSectionBuffer[i]=0;
        }
    }
//--- вернем значение prev_calculated для следующего вызова функции
    return(rates_total);
}
//+-----+

```

```

//| Изменяет цвет участков линии
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- количество цветов
int size=ArraySize(cols);
//---
string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- для каждого цветового индекса зададим новый цвет случайным образом
for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
{
//--- получим случайное число
int number=MathRand();
//--- получим индекс в массиве col[] как остаток от целочисленного деления
int i=number%size;
//--- установим цвет для каждого индекса как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0, // номер графического стиля
PLOT_LINE_COLOR, // идентификатор свойства
plot_color_ind, // индекс цвета, куда запишем цвет
cols[i]); // новый цвет

//--- запишем цвета
comm=comm+StringFormat("SectionColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString(plot_color_index));
ChartSetString(0,CHART_COMMENT,comm);
}

//--- 
}

//| Изменяет внешний вид отображаемой линии в индикаторе
//+-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах линии
string comm="";
//--- блок изменения толщины линии
int number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5; // толщина задается от 0 до 4
//--- установим цвет как свойство PLOT_LINE_WIDTH
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем толщину линии
comm=comm+" Width="+IntegerToString(width);

//--- блок изменения стиля линии
number=MathRand();
//--- делитель числа равен размеру массива styles
int size=ArraySize(styles);
//--- получим индекс для выбора нового стиля как остаток от целочисленного деления
int style_index=number%size;
}

```

```
//--- установим цвет как свойство PLOT_LINE_COLOR  
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);  
//--- запишем стиль линии  
comm=EnumToString(styles[style_index])+", "+comm;  
//--- выведем информацию на график через комментарий  
Comment(comm);  
}
```

## DRAW\_COLOR\_HISTOGRAM

Стиль DRAW\_COLOR\_HISTOGRAM рисует гистограмму цветными столбиками от нуля до указанного значения. Значения берутся из индикаторного буфера. Каждый столбик может иметь свой собственный цвет из заранее предопределенного набора цветов.

Толщину, цвет и стиль гистограммы можно задавать так же, как и для стиля [DRAW\\_HISTOGRAM - директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет изменять вид гистограммы в зависимости от текущей ситуации.

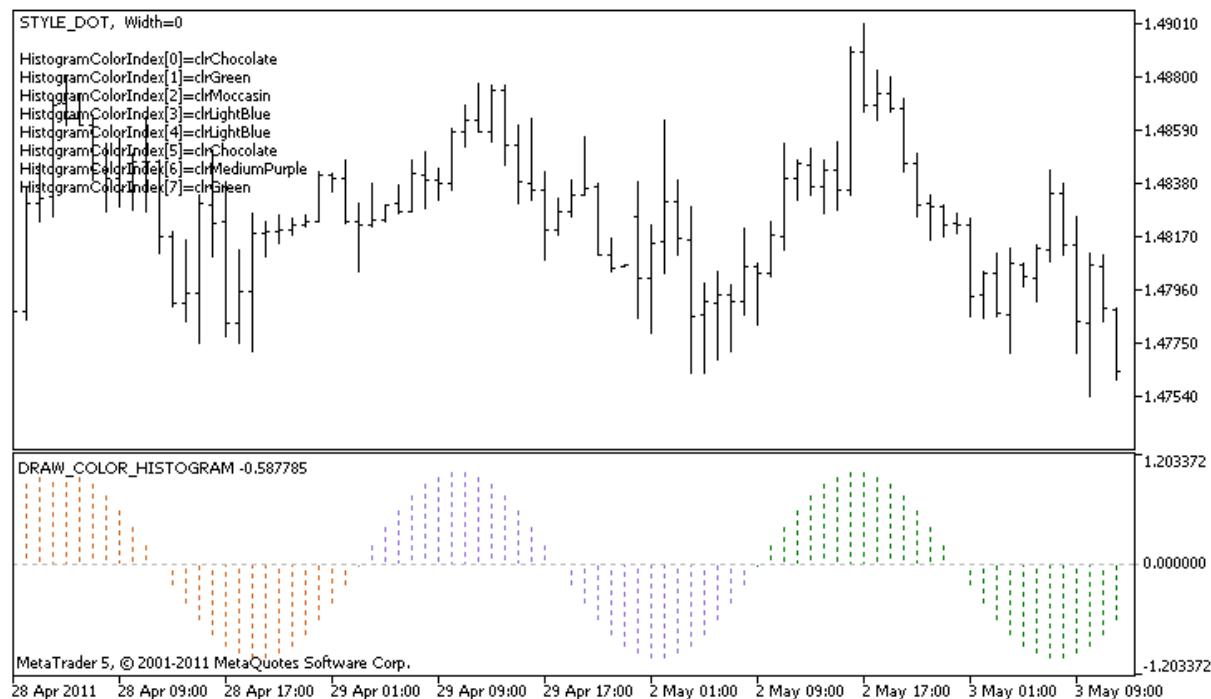
Так как на каждом баре рисуется столбик от нулевого уровня, то DRAW\_COLOR\_HISTOGRAM лучше использовать для отображения в отдельном подокне графика. Чаще всего этот тип графического построения используется для создания индикаторов осцилляторного типа, например, [Awesome Oscillator](#) или [Market Facilitation Index](#). Для пустых неотображаемых значений достаточно указывать нулевое значение.

Количество требуемых буферов для построения DRAW\_COLOR\_HISTOGRAM – 2:

- один буфер для хранения ненулевого значения вертикального отрезка на каждом баре, второй конец отрезка всегда находится на нулевой линии индикатора;
- один буфер для хранения индекса цвета, которым рисуется секция (имеет смысл задавать только для непустых значений).

Цвета можно задавать директивой компилятора `#property indicator_color1` через запятую. Количество цветов не может превышать 64.

Пример индикатора, рисующего заданным цветом синусоиду по функции [MathSin\(\)](#). Цвет, толщина и стиль **всех** столбиков гистограммы меняются случайным образом каждые N тиков. Параметр bars задает период синусоиды, то есть через заданное количество баров синусоида будет повторяться по циклу.



Обратите внимание, что первоначально для графического построения `plot1` со стилем `DRAW_COLOR_HISTOGRAM` задается 5 цветов с помощью директивы компилятора `#property indicator_color1`, а затем в функции `OnCalculate()` эти цвета выбираются случайным образом из 14 цветов, хранящихся в массиве `colors[]`. Параметр `N` вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```

//+-----+
//|                               DRAW_COLOR_HISTOGRAM.mq5 |
//|           Copyright 2011, MetaQuotes Software Corp. |
//|           https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_COLOR_HISTOGRAM"
#property description "Рисует синусоиду гистограммой в отдельном окне"
#property description "Цвет и толщина столбиков меняется случайным образом"
#property description "через каждые N тиков"
#property description "Параметр bars задает количество баров для повторяемости синусоиды"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots    1
//--- input параметры
input int      bars=30;          // период синусоиды в барах
input int      N=5;             // кол-во тиков для изменения гистограммы
//--- plot Color_Histogram
#property indicator_label1 "Color_Histogram"
#property indicator_type1 DRAW_COLOR_HISTOGRAM
//--- зададим 8 цветов для раскраски секций (они хранятся в специальном массиве)
#property indicator_color1 clrRed,clrGreen,clrBlue,clrYellow,clrMagenta,clrCyan,clrMediumPurple,clrIndigo
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- буфер значений
double         Color_HistogramBuffer[];
//--- буфер индексов цветов
double         Color_HistogramColors[];
//--- множитель для получения угла 2Pi в радианах при умножении на параметр bars
double         multiplier;
int            color_sections;
//--- массив для хранения цветов содержит 14 элементов
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurple,clrDarkKhaki
};
//--- массив для хранения стилей отрисовки линии
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOT2}

```

```

//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
    //--- indicator buffers mapping
    SetIndexBuffer(0,Color_HistogramBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,Color_HistogramColors,INDICATOR_COLOR_INDEX);
    //---- количество цветов для раскраски синусоиды
    color_sections=8;      // см. комментарий к свойству #property indicator_color1
    //--- вычислим множитель
    if(bars>1)multiplier=2.*M_PI/bars;
    else
    {
        PrintFormat("Задайте значение bars=%d больше 1",bars);
        //--- досрочное прекращение работы индикатора
        return(INIT_PARAMETERS_INCORRECT);
    }
    //---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function               |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=0;
    //--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;
    //--- если накопилось критическое число тиков
    if(ticks>=N)
    {
        //--- меняем свойства линии
        ChangeLineAppearance();
        //--- меняем цвета, которыми рисуется гистограмма
        ChangeColors(colors,color_sections);
        //--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }
}

```

```

//--- вычисления значений индикатора
int start=0;
//--- если расчет уже производился на предыдущем запуске OnCalculate
if(prev_calculated>0) start=prev_calculated-1; // установим начало расчетов с предыдущего запуска
//--- заполняем индикаторный буфер значениями
for(int i=start;i<rates_total;i++)
{
    //--- значение
    Color_HistogramBuffer[i]=sin(i*multiplier);
    //--- цвет
    int color_index=i%(bars*color_sections);
    color_index/=bars;
    Color_HistogramColors[i]=color_index;
}
//--- вернем значение prev_calculated для следующего вызова функции
return(rates_total);
}

//+-----+
//| Изменяет цвет участков линии |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- количество цветов
int size=ArraySize(cols);
//---
string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";
//--- для каждого цветового индекса зададим новый цвет случайнм образом
for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
{
    //--- получим случайное число
    int number=MathRand();
    //--- получим индекс в массиве col[] как остаток от целочисленного деления
    int i=number%size;
    //--- установим цвет для каждого индекса как свойство PLOT_LINE_COLOR
    PlotIndexSetInteger(0,          // номер графического стиля
                        PLOT_LINE_COLOR, // идентификатор свойства
                        plot_color_ind, // индекс цвета, куда запишем цвет
                        cols[i]);       // новый цвет
    //--- запишем цвета
    comm=comm+StringFormat("HistogramColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString(cols[i]));
    ChartSetString(0,CHART_COMMENT,comm);
}
//---
}

//+-----+
//| Изменяет внешний вид отображаемой линии в индикаторе |
//+-----+
void ChangeLineAppearance()

```

```
{  
//--- строка для формирования информации о свойствах линии  
string comm="";  
//--- блок изменения толщины линии  
int number=MathRand();  
//--- получим толщину как остаток от целочисленного деления  
int width=number%5; // толщина задается от 0 до 4  
//--- установим цвет как свойство PLOT_LINE_WIDTH  
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);  
//--- запишем толщину линии  
comm=comm+" Width="+IntegerToString(width);  
  
//--- блок изменения стиля линии  
number=MathRand();  
//--- делитель числа равен размеру массива styles  
int size=ArraySize(styles);  
//--- получим индекс для выбора нового стиля как остаток от целочисленного деления  
int style_index=number%size;  
//--- установим цвет как свойство PLOT_LINE_COLOR  
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);  
//--- запишем стиль линии  
comm=EnumToString(styles[style_index])+", "+comm;  
//--- выведем информацию на график через комментарий  
Comment(comm);  
}
```

## DRAW\_COLOR\_HISTOGRAM2

Стиль DRAW\_COLOR\_HISTOGRAM2 рисует заданным гистограмму - вертикальные отрезки по значениям двух индикаторных буферов. Но в отличие от одноцветного DRAW\_HISTOGRAM2 в этом стиле каждому столбiku гистограммы можно задать свой собственный цвет из предопределенного набора. Значения концов отрезков берутся из индикаторного буфера.

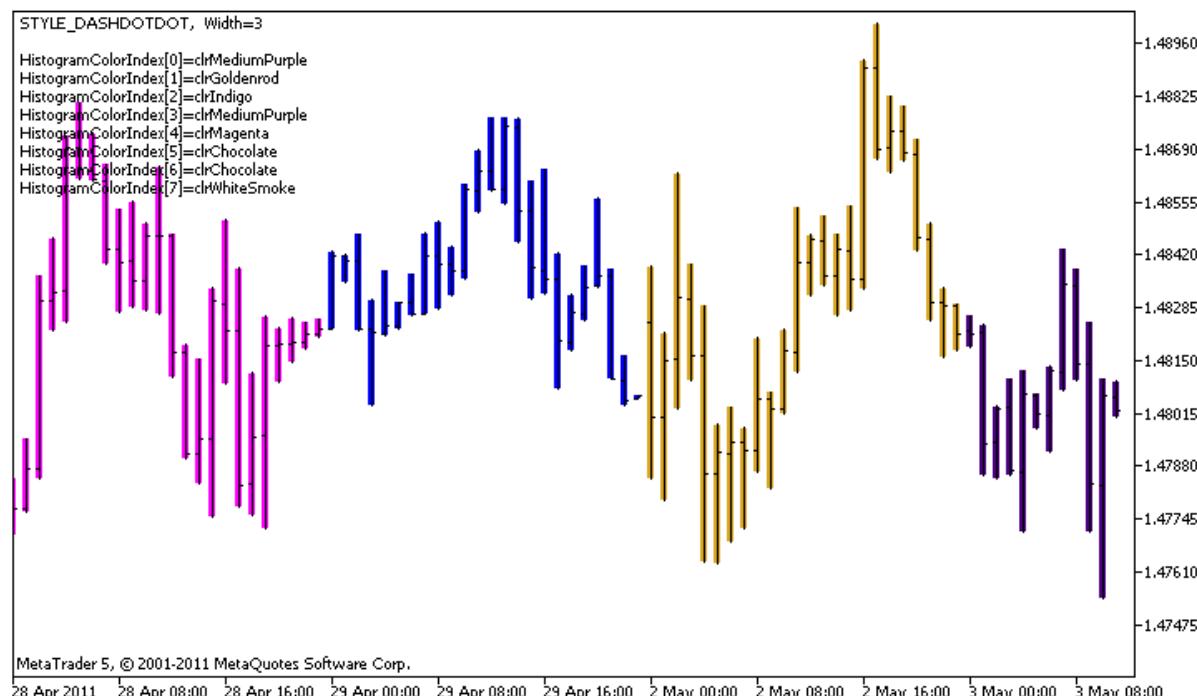
Толщину, стиль и цвета гистограммы можно задавать так же, как и для стиля [DRAW\\_HISTOGRAM2](#) - [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет изменять вид гистограммы в зависимости от текущей ситуации.

Стиль DRAW\_COLOR\_HISTOGRAM2 можно использовать как в отдельном подокне графика, так и в главном окне. Для пустых значений отрисовка не производится, все значения в индикаторных буферах нужно устанавливать явным образом. Инициализация буферов пустым значением не производится.

Количество требуемых буферов для построения DRAW\_COLOR\_HISTOGRAM2 – 3:

- два буфера для хранения верхнего и нижнего конца вертикального отрезка на каждом баре;
- один буфер для хранения индекса цвета, которым рисуется отрезок (имеет смысл задавать только для непустых значений).

Пример индикатора, рисующего заданным цветом гистограмму между ценами High и Low. Для каждого дня недели линии гистограммы рисуются своим цветом. Цвет каждого дня, толщина и стиль гистограммы меняются случайным образом каждые N тиков.



Обратите внимание, что первоначально для графического построения `plot1` со стилем DRAW\_COLOR\_HISTOGRAM2 задается 5 цветов с помощью директивы компилятора [#property indicator\\_color1](#), а затем в функции [OnCalculate\(\)](#) цвета выбираются случайным образом из 14 цветов, хранящихся в массиве `colors[]`.

Параметр N вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```

//+-----+
//|                               DRAW_COLOR_HISTOGRAM2.mq5  |
//|           Copyright 2011, MetaQuotes Software Corp.  |
//|                           https://www.mql5.com  |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_COLOR_HISTOGRAM2"
#property description "Рисует на каждом баре отрезок между Open и Close"
#property description "Цвет, толщина и стиль меняется случайным образом"
#property description "через каждые N тиков"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots 1
//--- plot ColorHistogram_2
#property indicator_label1 "ColorHistogram_2"
#property indicator_type1 DRAW_COLOR_HISTOGRAM2
//--- зададим 5 цветов для раскраски гистограммы по дням недели (они хранятся в специальном массиве)
#property indicator_color1 clrRed,clrBlue,clrGreen,clrYellow,clrMagenta
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1

//--- input параметр
input int      N=5;                      // кол-во тиков для изменения гистограммы
int            color_sections;
//--- буфера значений
double         ColorHistogram_2Buffer1[];
double         ColorHistogram_2Buffer2[];
//--- буфер индексов цвета
double         ColorHistogram_2Colors[];
//--- массив для хранения цветов содержит 14 элементов
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurple
};
//--- массив для хранения стилей отрисовки линии
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT};
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{

```

```

//--- indicator buffers mapping
SetIndexBuffer(0,ColorHistogram_2Buffer1,INDICATOR_DATA);
SetIndexBuffer(1,ColorHistogram_2Buffer2,INDICATOR_DATA);
SetIndexBuffer(2,ColorHistogram_2Colors,INDICATOR_COLOR_INDEX);
//--- установим пустое значение
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- количество цветов для раскраски синусоиды
color_sections=8; // см. комментарий к свойству #property indicator_color1
//---
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=0;
//--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;
//--- если накопилось критическое число тиков
    if(ticks>=N)
    {
        //--- меняем свойства линии
        ChangeLineAppearance();
        //--- меняем цвета, которыми рисуются гистограмма
        ChangeColors(colors,color_sections);
        //--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }

//--- вычисления значений индикатора
    int start=0;
//--- для получения дня недели по времени открытия каждого бара
    MqlDateTime dt;
//--- если расчет уже производился на предыдущем запуске OnCalculate
    if(prev_calculated>0) start=prev_calculated-1; // установим начало расчетов с предыдущего запуска
//--- заполняем индикаторный буфер значениями
    for(int i=start;i<rates_total;i++)
    {
        TimeToStruct(time[i],dt);
    }
}

```

```

//--- значение
ColorHistogram_2Buffer1[i]=high[i];
ColorHistogram_2Buffer2[i]=low[i];
//--- зададим индекс цвета по дню недели
int day=dt.day_of_week;
ColorHistogram_2Colors[i]=day;
}

//--- вернем значение prev_calculated для следующего вызова функции
return(rates_total);
}

//+-----+
//| Изменяет цвет участков линии |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- количество цветов
int size=ArraySize(cols);
//---
string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- для каждого цветового индекса зададим новый цвет случайным образом
for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
{
//--- получим случайное число
int number=MathRand();
//--- получим индекс в массиве col[] как остаток от целочисленного деления
int i=number%size;
//--- установим цвет для каждого индекса как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,           // номер графического стиля
                    PLOT_LINE_COLOR, // идентификатор свойства
                    plot_color_ind, // индекс цвета, куда запишем цвет
                    cols[i]);       // новый цвет

//--- запишем цвета
comm=comm+StringFormat("HistogramColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString(ChartSetString(0,CHART_COMMENT,comm)));
}

//---
}

//+-----+
//| Изменяет внешний вид отображаемой линии в индикаторе |
//+-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах линии
string comm="";
//--- блок изменения толщины линии
int number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5; // толщина задается от 0 до 4
}

```

```
//--- установим цвет как свойство PLOT_LINE_WIDTH
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем толщину линии
comm=comm+" Width="+IntegerToString(width);

//--- блок изменения стиля линии
number=MathRand();
//--- делитель числа равен размеру массива styles
int size=ArraySize(styles);
//--- получим индекс для выбора нового стиля как остаток от целочисленного деления
int style_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- запишем стиль линии
comm=EnumToString(styles[style_index])+", "+comm;
//--- выведем информацию на график через комментарий
Comment(comm);
}
```

## DRAW\_COLOR\_ARROW

Стиль DRAW\_COLOR\_ARROW рисует на графике цветом стрелки (символы из набора [Wingdings](#)) по значению индикаторного буфера. В отличие от стиля DRAW\_ARROW, в нем можно для каждого символа задавать цвет из предопределенного набора цветов, задаваемых свойством [indicator\\_color1](#).

Толщину и цвет символов можно задавать так же, как и для стиля [DRAW\\_ARROW](#) - [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет изменять вид индикатора в зависимости от текущей ситуации.

Код символа для вывода на график задается с помощью свойства [PLOT\\_ARROW](#).

```
//--- зададим код символа из шрифта Wingdings для отрисовки в PLOT_ARROW
PlotIndexSetInteger(0, PLOT_ARROW, code);
```

По умолчанию значение PLOT\_ARROW=159 (кружок).

Каждая стрелка фактически представляет собой символ, который имеет высоту и точку привязки, и может закрывать собою некоторую важную информацию на графике (например, цену закрытия на баре). Поэтому можно дополнительно указать вертикальное смещение в пикселях, которое не зависит от масштаба графика. На это указанное количество пикселей стрелки будут визуально смещены по вертикали, хотя сами значения индикатора останутся те же самые:

```
//--- зададим смещение стрелок по вертикали в пикселях
PlotIndexSetInteger(0, PLOT_ARROW_SHIFT, shift);
```

Отрицательное значение PLOT\_ARROW\_SHIFT означает смещение стрелок вверх, положительно значения смещает стрелки вниз.

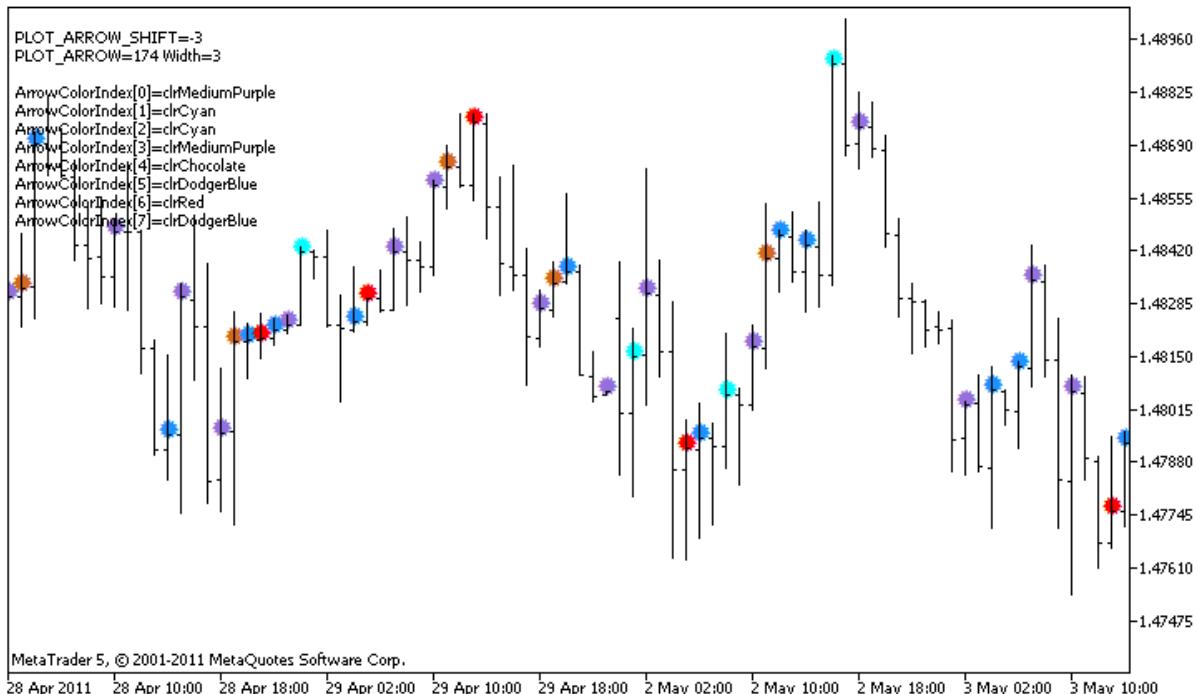
Стиль DRAW\_COLOR\_ARROW можно использовать как в отдельном подокне графика, так и в главном окне. Пустые значения не отрисовываются и не отображаются в "Окне данных", все значения в индикаторных буферах нужно устанавливать явным образом. Инициализация буферов пустым значением не производится.

```
//--- установим пустое значение
PlotIndexSetDouble(индекс_построения_DRAW_COLOR_ARROW, PLOT_EMPTY_VALUE, 0);
```

Количество требуемых буферов для построения DRAW\_COLOR\_ARROW – 2:

- один буфер для хранения значения цены, по которой рисуется символ (плюс смещение в пикселях, задаваемое свойством PLOT\_ARROW\_SHIFT);
- один буфер для хранения индекса цвета, которым рисуется стрелка (имеет смысл задавать только для непустых значений).

Пример индикатора, рисующего стрелки на каждом баре, у которого цена закрытия Close больше цены закрытия предыдущего бара. Толщина, смещение и код символа **всех** стрелок меняются случайным образом каждые N тиков. Цвет символа зависит от номера бара, на котором он нарисован.



В примере первоначально для графического построения `plot1` со стилем `DRAW_COLOR_ARROW` свойства, цвет и размер задаются с помощью директивы компилятора `#property`, а затем в функции `OnCalculate()` свойства случайным образом меняются. Параметр `N` вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

Обратите внимание, первоначально задается 8 цветов с помощью директивы компилятора `#property`, а затем в функции `OnCalculate()` цвет выбирается случайным образом из 14 цветов, хранящихся в массиве `colors[]`.

```
//+-----+
//|                                              DRAW_COLOR_ARROW.mq5 |
//|          Copyright 2011, MetaQuotes Software Corp. |
//|          https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_COLOR_ARROW"
#property description "Рисует на графике разным цветом стрелки, задаваемые символами"
#property description "Цвет, размер, смещение и код символа стрелки меняется"
#property description " случайным образом через каждые N тиков"
#property description "Параметр code задает базовое значение: код=159 (кружок)"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots    1
//--- plot ColorArrow
#property indicator_label1 "ColorArrow"
```

```

#property indicator_type1 DRAW_COLOR_ARROW
//--- зададим 8 цветов для раскраски гистограммы (они хранятся в специальном массиве)
#property indicator_color1 clrRed,clrBlue,clrSeaGreen,clrGold,clrDarkOrange,clrMagenta
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1

//--- input параметры
input int      N=5;           // кол-во тиков для изменения
input ushort   code=159;       // код символа для отрисовки в DRAW_ARROW
int          color_sections;
//--- индикаторный буфер для построения
double        ColorArrowBuffer[];
//--- буфер для хранения индексов цвета
double        ColorArrowColors[];
//--- массив для хранения цветов содержит 14 элементов
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPurple
};

//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,ColorArrowBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,ColorArrowColors,INDICATOR_COLOR_INDEX);
//--- зададим код символа для отрисовки в PLOT_ARROW
    PlotIndexSetInteger(0,PLOT_ARROW,code);
//--- зададим смещение стрелок по вертикали в пикселях
    PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,5);
//--- установим в качестве пустого значения 0
    PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- количество цветов для раскраски синусоиды
    color_sections=8; // см. комментарий к свойству #property indicator_color1
//---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
```

```

        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
        static int ticks=0;
//--- считаем тики для изменения цвета, размера, смещения и кода стрелки
        ticks++;
//--- если накопилось критическое число тиков
        if(ticks>=N)
        {
//--- меняем свойства стрелок
            ChangeLineAppearance();
//--- меняем цвета, которыми рисуются гистограмма
            ChangeColors(colors,color_sections);
//--- сбрасываем счетчик тиков в ноль
            ticks=0;
        }

//--- блок расчета значений индикатора
        int start=1;
        if(prev_calculated>0) start=prev_calculated-1;
//--- цикл расчета
        for(int i=1;i<rates_total;i++)
        {
//--- если текущая цена Close больше предыдущей, ставим стрелку
            if(close[i]>close[i-1])
                ColorArrowBuffer[i]=close[i];
//--- в противном случае указываем нулевое значение
            else
                ColorArrowBuffer[i]=0;
//--- цвет стрелки
            int index=i%color_sections;
            ColorArrowColors[i]=index;
        }
//--- return value of prev_calculated for next call
        return(rates_total);
    }
//+-----+
//| Изменяет цвет участков линии |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- количество цветов
    int size=ArraySize(cols);
//---
    string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";
//--- для каждого цветового индекса зададим новый цвет случайнм образом
    for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
}

```

```

{
    //--- получим случайное число
    int number=MathRand();
    //--- получим индекс в массиве col[] как остаток от целочисленного деления
    int i=number%size;
    //--- установим цвет для каждого индекса как свойство PLOT_LINE_COLOR
    PlotIndexSetInteger(0,           // номер графического стиля
                        PLOT_LINE_COLOR, // идентификатор свойства
                        plot_color_ind, // индекс цвета, куда запишем цвет
                        cols[i]);       // новый цвет

    //--- запишем цвета
    comm=comm+StringFormat("ArrowColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString());
    ChartSetString(0,CHART_COMMENT,comm);
}

//---

}

//+-----+
//| Изменяет внешний вид отображаемой линии в индикаторе |
//+-----+
void ChangeLineAppearance()
{
    //--- строка для формирования информации о свойствах линии
    string comm="";
    //--- блок изменения толщины линии
    int number=MathRand();
    //--- получим толщину как остаток от целочисленного деления
    int width=number%5; // толщина задается от 0 до 4
    //--- установим цвет как свойство PLOT_LINE_WIDTH
    PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
    //--- запишем толщину линии
    comm=comm+" Width="+IntegerToString(width);

    //--- блок изменения кода стрелки (PLOT_ARROW)
    number=MathRand();
    //--- получим остаток от целочисленного деления для вычисления нового кода стрелки (от
    int code_add=number%20;
    //--- установим новый код символа как сумму code+code_add
    PlotIndexSetInteger(0,PLOT_ARROW,code+code_add);
    //--- запишем код символа PLOT_ARROW
    comm="\r\n"+PLOT_ARROW=IntegerToString(code+code_add)+comm;

    //--- блок изменения смещения стрелок по вертикали в пикселях
    number=MathRand();
    //--- получим смещение как остаток от целочисленного деления
    int shift=20-number%41;
    //--- установим новое смещение
    PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
    //--- запишем смещение PLOT_ARROW_SHIFT
    comm="\r\n"+PLOT_ARROW_SHIFT=IntegerToString(shift)+comm;
}

```

```
//--- выводим информацию на график через комментарий  
Comment(comm);  
}
```

## DRAW\_COLOR\_ZIGZAG

Стиль DRAW\_COLOR\_ZIGZAG рисует отрезки разного цвета по значениям двух индикаторных буферов. Этот стиль является цветной версией стиля [DRAW\\_ZIGZAG](#), то есть позволяет задавать каждому отрезку свой собственный цвет из заранее предопределенного набора цветов. Отрезки рисуются от значения в первом буфере до значения во втором индикаторном буфере. Ни один из буферов не может содержать только пустые значения, так как в этом случае отрисовка не происходит.

Толщину, цвет и стиль отображения линии можно задавать так же, как и для стиля [DRAW\\_ZIGZAG](#) - [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет "оживить" индикаторы, чтобы они меняли свой вид в зависимости от текущей ситуации.

Секции рисуются от одного непустого значения одного буфера до непустого значения другого индикаторного буфера. Чтобы указать, какое значение следует считать "пустым", установите это значение в свойстве [PLOT\\_EMPTY\\_VALUE](#):

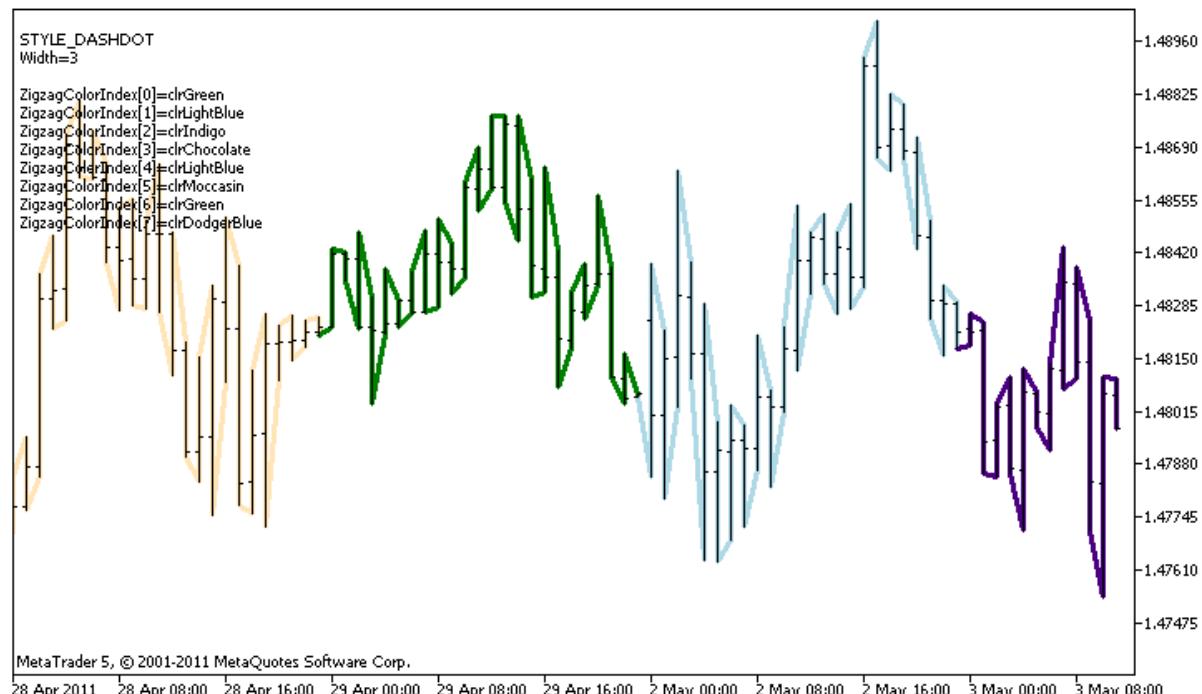
```
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
PlotIndexSetDouble(индекс_построения_DRAW_COLOR_ZIGZAG, PLOT_EMPTY_VALUE, 0);
```

Всегда явно заполняйте значениями индикаторные буфера, на пропускаемых барах указывайте в буфере пустое значение.

Количество требуемых буферов для построения DRAW\_COLOR\_ZIGZAG – 3:

- два буфера для хранения значений концов секций зигзага;
- один буфер для хранения индекса цвета, которым рисуется секция (имеет смысл задавать только для непустых значений).

Пример индикатора, рисующего пилю по ценам High и Low. Цвет, толщина и стиль линий зигзага меняются случайным образом каждые N тиков.



Обратите внимание, первоначально для графического построения `plot1` со стилем `DRAW_COLOR_ZIGZAG` задается 8 цветов с помощью директивы компилятора `#property`, а затем в функции `OnCalculate()` цвет выбирается случайным образом из 14 цветов, хранящихся в массиве `colors[]`.

Параметр N вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).

```

//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,Color_ZigzagBuffer1,INDICATOR_DATA);
    SetIndexBuffer(1,Color_ZigzagBuffer2,INDICATOR_DATA);
    SetIndexBuffer(2,Color_ZigzagColors,INDICATOR_COLOR_INDEX);
//--- количество цветов для раскраски зигзага
    color_sections=8; // см. комментарий к свойству #property indicator_color1
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
//--- считаем тики для изменения стиля, цвета и толщины линии
    ticks++;
//--- если накопилось достаточное число тиков
    if(ticks>=N)
    {
//--- меняем свойства линии
        ChangeLineAppearance();
//--- меняем цвета, которыми рисуются секции
        ChangeColors(colors,color_sections);
//--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }

//--- структура времени понадобится для получения дня недели каждого бара
    MqlDateTime dt;

//--- позиция начала расчетов
    int start=0;
//--- если индикатор рассчитывался на предыдущем тике, то начинам расчет с предпоследн
    if(prev_calculated!=0) start=prev_calculated-1;
//--- цикл расчетов
}

```

```

for(int i=start;i<rates_total;i++)
{
    //--- запишем время открытия бара в структуру
    TimeToStruct(time[i],dt);

    //--- если номер бара четный
    if(i%2==0)
    {
        //--- пишем в 1-ый буфер High, во 2-ой Low
        Color_ZigzagBuffer1[i]=high[i];
        Color_ZigzagBuffer2[i]=low[i];
        //--- цвет отрезка
        Color_ZigzagColors[i]=dt.day_of_year%color_sections;
    }
    //--- номер бара нечетный
    else
    {
        //--- заполняем бар в обратном порядке
        Color_ZigzagBuffer1[i]=low[i];
        Color_ZigzagBuffer2[i]=high[i];
        //--- цвет отрезка
        Color_ZigzagColors[i]=dt.day_of_year%color_sections;
    }
}
//--- return value of prev_calculated for next call
return(rates_total);
}

//+-----+
//| Изменяет цвет отрезков зигзага |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- количество цветов
int size=ArraySize(cols);
//---
string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- для каждого цветового индекса зададим новый цвет случайным образом
for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
{
    //--- получим случайное число
    int number=MathRand();
    //--- получим индекс в массиве col[] как остаток от целочисленного деления
    int i=number%size;
    //--- установим цвет для каждого индекса как свойство PLOT_LINE_COLOR
    PlotIndexSetInteger(0,           // номер графического стиля
                        PLOT_LINE_COLOR, // идентификатор свойства
                        plot_color_ind, // индекс цвета, куда запишем цвет
                        cols[i]);       // новый цвет
}
}

```

```

//--- запишем цвета
comm=comm+StringFormat("ZigzagColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString(ChartSetString(0,CHART_COMMENT,comm));
}

//---
}

//+-----+
// | Изменяет внешний вид отрезков в зигзаге |
//+-----+

void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах Color_ZigZag
string comm="";
//--- блок изменения толщины линии
int number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5; // толщина задается от 0 до 4
//--- установим цвет как свойство PLOT_LINE_WIDTH
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем толщину линии
comm=comm+"\r\nWidth="+IntegerToString(width);

//--- блок изменения стиля линии
number=MathRand();
//--- делитель числа равен размеру массива styles
int size=ArraySize(styles);
//--- получим индекс для выбора нового стиля как остаток от целочисленного деления
int style_index=number%size;
//--- установим цвет как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- запишем стиль линии
comm="\r\n"+EnumToString(styles[style_index])+" "+comm;
//--- выведем информацию на график через комментарий
Comment(comm);
}

```

## DRAW\_COLOR\_BARS

Стиль DRAW\_COLOR\_BARS рисует бары по значениям четырех индикаторных буферов, в которых содержатся цены Open, High, Low и Close. Этот стиль является продвинутой версией стиля [DRAW\\_BARS](#) и позволяет задавать для каждого бара свой цвет из заранее предопределенного набора цветов. Предназначен для создания собственных индикаторов в виде баров, в том числе в отдельном подокне графика и по другим финансовым инструментам.

Цвет баров можно задавать [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет "оживить" индикаторы, чтобы они меняли свой вид в зависимости от текущей ситуации.

Индикатор рисуется только для тех баров, для которых заданы непустые значения **всех** четырех индикаторных буферов. Чтобы указать, какое значение следует считать "пустым", установите это значение в свойстве [PLOT\\_EMPTY\\_VALUE](#):

```
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
PlotIndexSetDouble(индекс_построения_DRAW_COLOR_BARS, PLOT_EMPTY_VALUE, 0);
```

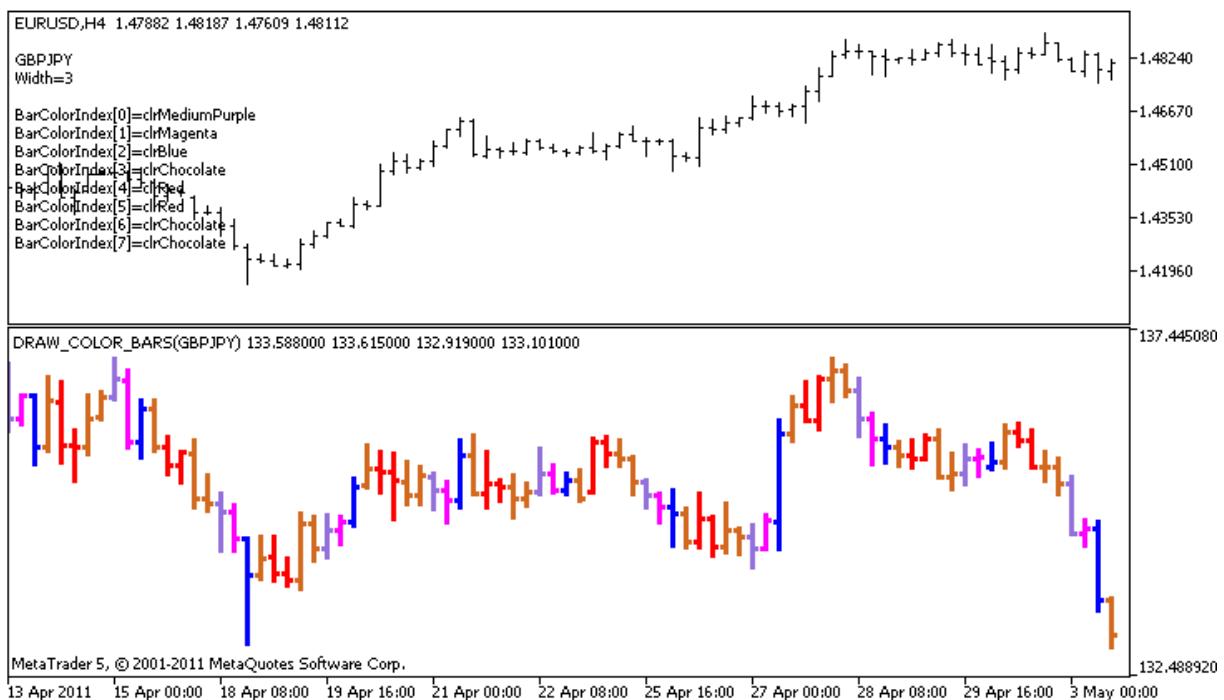
Всегда явно заполняйте значениями индикаторные буфера, на пропускаемых барах указывайте в буфере пустое значение.

Количество требуемых буферов для построения DRAW\_COLOR\_BARS – 5:

- четыре буфера для хранения значений Open, High, Low и Close;
- один буфер для хранения индекса цвета, которым рисуется бар (имеет смысл задавать только отрисовываемых баров).

Все буфера для построения должны идти последовательно один за другим в заданном порядке: Open, High, Low, Close и буфер цвета. Ни один из ценовых буферов не может содержать только пустые значения, так как в этом случае отрисовка не происходит.

Пример индикатора, рисующего в отдельном окне бары по указанному финансовому инструменту. Цвет баров меняется случайным образом каждые **N** тиков. Параметр **N** вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).



Обратите внимание, первоначально для графического построения `plot1` со стилем `DRAW_COLOR_BARS` задается 8 цветов с помощью директивы компилятора `#property`, а затем в функции `OnCalculate()` цвет выбирается случайным образом из 14 цветов, хранящихся в массиве `colors[]`.

```
//+-----+
//|                               DRAW_COLOR_BARS.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_COLOR_BARS"
#property description "Рисует в отдельном окне разным цветом бары по выбранному символу"
#property description "Цвет и толщина баров, а также символ, меняются случайным образом через каждые N тиков"

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots   1
//--- plot ColorBars
#property indicator_label1 "ColorBars"
#property indicator_type1  DRAW_COLOR_BARS
//--- зададим 8 цветов для раскраски баров (они хранятся в специальном массиве)
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input параметры
```

```

input int      N=5;                      // количество тиков для смены вида
input int      bars=500;                  // сколько баров показывать
input bool     messages=false;           // вывод сообщений в лог "Эксперты"
//--- индикаторные буферы
double        ColorBarsBuffer1[];
double        ColorBarsBuffer2[];
double        ColorBarsBuffer3[];
double        ColorBarsBuffer4[];
double        ColorBarsColors[];
//--- имя символа
string        symbol;
int          bars_colors;
//--- массив для хранения цветов содержит 14 элементов
color         colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrMagenta,clrCyan,clrMediumPurple
};

//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,ColorBarsBuffer1,INDICATOR_DATA);
    SetIndexBuffer(1,ColorBarsBuffer2,INDICATOR_DATA);
    SetIndexBuffer(2,ColorBarsBuffer3,INDICATOR_DATA);
    SetIndexBuffer(3,ColorBarsBuffer4,INDICATOR_DATA);
    SetIndexBuffer(4,ColorBarsColors,INDICATOR_COLOR_INDEX);

//---- количество цветов для раскраски баров
    bars_colors=8;    // см. комментарий к свойству #property indicator_color1
//---
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function                |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
    static int ticks=0;
}

```

```

//--- считаем тики для изменения стиля, цвета и толщины бара
ticks++;
//--- если накопилось достаточное число тиков
if(ticks>=N)
{
    //--- выберем новый символ из окна "Обзор рынка"
symbol=GetRandomSymbolName();
//--- меняем свойства линии
ChangeLineAppearance();
//--- меняем цвета, которыми рисуются бары
ChangeColors(colors,bars_colors);
int tries=0;
//--- сделаем 5 попыток заполнить буфера ценами из symbol
while(!CopyFromSymbolToBuffers(symbol,rates_total,bars_colors) && tries<5)
{
    //--- счетчик вызовов функции CopyFromSymbolToBuffers()
    tries++;
}
//--- сбрасываем счетчик тиков в ноль
ticks=0;
}
//--- return value of prev_calculated for next call
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буферы ценами |+
//+-----+
bool CopyFromSymbolToBuffers(string name,int total,int bar_colors)
{
//--- в массив rates[] будем копировать цены Open, High, Low и Close
MqlRates rates[];
//--- счетчик попыток
int attempts=0;
//--- сколько скопировано
int copied=0;
//--- делаем 25 попыток получить таймсерию по нужному символу
while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
{
    Sleep(100);
    attempts++;
    if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attempts);
}
//--- если не удалось скопировать достаточно большое количество баров
if(copied!=bars)
{
    //--- сформируем строку сообщения
string comm=StringFormat("Для символа %s удалось получить только %d баров из %d
                                name,
                                copied,

```

```

        bars
    );
//--- выведем сообщение в комментарий на главное окно графика
Comment(comm);
//--- выводим сообщения
if(messages) Print(comm);
return(false);
}
else
{
//--- установим отображение символа
PlotIndexSetString(0,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;"+name+
IndicatorSetString(INDEX_SHORTNAME,"DRAW_COLOR_BARS("+name+ ")");
}
//--- инициализируем буферы пустыми значениями
ArrayInitialize(ColorBarsBuffer1,0.0);
ArrayInitialize(ColorBarsBuffer2,0.0);
ArrayInitialize(ColorBarsBuffer3,0.0);
ArrayInitialize(ColorBarsBuffer4,0.0);

//--- копируем цены в буфера
for(int i=0;i<copied;i++)
{
//--- вычислим соответствующий индекс для буферов
int buffer_index=total-copied+i;
//--- записываем цены в буфера
ColorBarsBuffer1[buffer_index]=rates[i].open;
ColorBarsBuffer2[buffer_index]=rates[i].high;
ColorBarsBuffer3[buffer_index]=rates[i].low;
ColorBarsBuffer4[buffer_index]=rates[i].close;
//---
ColorBarsColors[buffer_index]=i%bar_colors;
}
return(true);
}
//-----+
//| Возвращает случайным образом символ из Market Watch |
//-----+
string GetRandomSymbolName()
{
//--- количество символов, показываемых в окне "Обзор рынка"
int symbols=SymbolsTotal(true);
//--- позиция символа в списке - случайное число от 0 до symbols
int number=MathRand()%symbols;
//--- вернем имя символа по указанной позиции
return SymbolName(number,true);
}
//-----+
//| Изменяет цвет отрезков зигзага |

```

```

//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- количество цветов
int size=ArraySize(cols);
//---

string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- для каждого цветового индекса зададим новый цвет случайным образом
for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
{
//--- получим случайное число
int number=MathRand();
//--- получим индекс в массиве col[] как остаток от целочисленного деления
int i=number%size;
//--- установим цвет для каждого индекса как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0,                      // номер графического стиля
                    PLOT_LINE_COLOR,      // идентификатор свойства
                    plot_color_ind,       // индекс цвета, куда запишем цвет
                    cols[i]);            // новый цвет

//--- запишем цвета
comm=comm+StringFormat("BarColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString
ChartSetString(0,CHART_COMMENT,comm);
}

//---+
//| Изменяет внешний вид баров |
//+-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах баров
string comm="";

//--- блок изменения толщины баров
int number=MathRand();
//--- получим толщину как остаток от целочисленного деления
int width=number%5;    // толщина задается от 0 до 4
//--- установим цвет как свойство PLOT_LINE_WIDTH
PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- запишем толщину линии
comm=comm+"\r\nWidth="+IntegerToString(width);

//--- запишем имя символа
comm="\r\n"+symbol+comm;

//--- выведем информацию на график через комментарий
Comment(comm);
}

```



## DRAW\_COLOR\_CANDLES

Стиль DRAW\_COLOR\_CANDLES как и [DRAW\\_CANDLES](#) рисует японские свечи по значениям четырех индикаторных буферов, в которых содержатся цены Open, High, Low и Close. Но кроме этого он позволяет задавать цвет для каждой свечи из заданного набора. Для этого в стиль добавлен специальный цветовой буфер, который хранит индексы цветов для каждого бара. Предназначен для создания собственных индикаторов в виде свечей, в том числе в отдельном подокне графика и по другим финансовым инструментам.

Количество цветов для раскраски свечей можно задавать [директивами компилятора](#) или динамически с помощью функции [PlotIndexSetInteger\(\)](#). Динамическое изменение свойств графического построения позволяет "оживить" индикаторы, чтобы они меняли свой вид в зависимости от текущей ситуации.

Индикатор рисуется только для тех баров, для которых заданы непустые значения четырех индикаторных буферов для хранения цен. Чтобы указать, какое значение следует считать "пустым", установите это значение в свойстве [PLOT\\_EMPTY\\_VALUE](#):

```
//--- значение 0 (пустое значение) не будет участвовать в отрисовке
PlotIndexSetDouble(индекс_построения_DRAW_COLOR_CANDLES, PLOT_EMPTY_VALUE, 0);
```

Всегда явно заполняйте значениями индикаторные буфера, на пропускаемых барах указывайте в буфере пустое значение.

Количество требуемых буферов для построения DRAW\_COLOR\_CANDLES – 5:

- четыре буфера для хранения значений Open, High, Low и Close;
- один буфер для хранения индекса цвета, которым рисуется свеча (имеет смысл задавать только отрисовываемых свечей).

Все буфера для построения должны идти последовательно один за другим в заданном порядке: Open, High, Low, Close и буфер цвета. Ни один из ценовых буферов не может содержать только пустые значения, так как в этом случае отрисовка не происходит.

Пример индикатора, рисующего в отдельном окне японские свечи по указанному финансовому инструменту. Цвет свечей меняется случайным образом каждые N тиков. Параметр N вынесен во [внешние параметры](#) индикатора для возможности ручной установки (закладка "Параметры" в окне свойств индикатора).



Обратите внимание, первоначально для графического построения `plot1` цвет задается с помощью директивы компилятора `#property`, а затем в функции `OnCalculate()` выбирается новый цвет случайным образом из заранее подготовленного списка.

```
//+-----+
//|                               DRAW_COLOR_CANDLES.mq5 |
//|           Copyright 2011, MetaQuotes Software Corp. |
//|           https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "Индикатор для демонстрации DRAW_COLOR_CANDLES."
#property description "Рисует в отдельном окне разным цветом свечи по случайно выбранн"
#property description " "
#property description "Цвет и толщина свечей, а также символ, меняются"
#property description "случайным образом через каждые N тиков."

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots    1
//--- plot ColorCandles
#property indicator_label1  "ColorCandles"
#property indicator_type1   DRAW_COLOR_CANDLES
//--- зададим 8 цветов для раскраски свечей (они хранятся в специальном массиве)
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

```

//--- input параметры
input int      N=5;                      // количество тиков для смены вида
input int      bars=500;                   // сколько свечей показывать
input bool     messages=false;           // вывод сообщений в лог "Эксперты"
//--- индикаторные буферы
double        ColorCandlesBuffer1[];
double        ColorCandlesBuffer2[];
double        ColorCandlesBuffer3[];
double        ColorCandlesBuffer4[];
double        ColorCandlesColors[];
int          candles_colors;
//--- имя символа
string        symbol;
//--- массив для хранения цветов содержит 14 элементов
color colors[]=
{
    clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
    clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrMagenta,clrCyan,clrMediumPurple
};

//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- если bars слишком мало - досрочно завершаем работу
if(bars<50)
{
    Comment("Укажите большее количество баров! Работа индикатора прекращена");
    return(INIT_PARAMETERS_INCORRECT);
}
//--- indicator buffers mapping
SetIndexBuffer(0,ColorCandlesBuffer1,INDICATOR_DATA);
SetIndexBuffer(1,ColorCandlesBuffer2,INDICATOR_DATA);
SetIndexBuffer(2,ColorCandlesBuffer3,INDICATOR_DATA);
SetIndexBuffer(3,ColorCandlesBuffer4,INDICATOR_DATA);
SetIndexBuffer(4,ColorCandlesColors,INDICATOR_COLOR_INDEX);
//--- пустое значение
PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- имя символа, по которому рисуются бары
symbol=_Symbol;
//--- установим отображение символа
PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+symbol);
IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_CANDLES("+symbol+")");
//--- количество цветов для раскраски свечей
candles_colors=8;           // см. комментарий к свойству #property indicator_color1
//---
return(INIT_SUCCEEDED);
}
//+-----+

```

```

//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int ticks=INT_MAX-100;
    //--- считаем тики для изменения стиля и цвета
    ticks++;
    //--- если накопилось достаточное число тиков
    if(ticks>=N)
    {
        //--- выберем новый символ из окна "Обзор рынка"
        symbol=GetRandomSymbolName();
        //--- сменим вид
        ChangeLineAppearance();
        //--- меняем цвета, которыми рисуются бары
        ChangeColors(colors,candles_colors);

        int tries=0;
        //--- сделаем 5 попыток заполнить буферы plot1 ценами из symbol
        while(!CopyFromSymbolToBuffers(symbol,rates_total,0,
                                       ColorCandlesBuffer1,ColorCandlesBuffer2,ColorCandlesBuffer3,
                                       ColorCandlesBuffer4,ColorCandlesColors,candles_colors)
              && tries<5)
        {
            //--- счетчик вызовов функции CopyFromSymbolToBuffers()
            tries++;
        }
        //--- сбрасываем счетчик тиков в ноль
        ticks=0;
    }
    //--- return value of prev_calculated for next call
    return(rates_total);
}
//| Заполняет указанную свечу
//+-----+
bool CopyFromSymbolToBuffers(string name,
                             int total,
                             int plot_index,
                             double &buff1[],
```

```

        double &buff2[],
        double &buff3[],
        double &buff4[],
        double &col_buffer[],
        int     cndl_colors
    )
}

//--- в массив rates[] будем копировать цены Open, High, Low и Close
MqlRates rates[];
//--- счетчик попыток
int attempts=0;
//--- сколько скопировано
int copied=0;
//--- делаем 25 попыток получить таймсерию по нужному символу
while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
{
    Sleep(100);
    attempts++;
    if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attempts);
}
//--- если не удалось скопировать достаточно большое количество баров
if(copied!=bars)
{
    //--- сформируем строку сообщения
    string comm=StringFormat("Для символа %s удалось получить только %d баров из %d",
                             name,
                             copied,
                             bars
                            );
    //--- выведем сообщение в комментарий на главное окно графика
    Comment(comm);
    //--- выводим сообщения
    if(messages) Print(comm);
    return(false);
}
else
{
    //--- установим отображение символа
    PlotIndexSetString(plot_index,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;
    IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_CANDLES("+symbol+ ")");
}
//--- инициализируем буфера пустыми значениями
ArrayInitialize(buff1,0.0);
ArrayInitialize(buff2,0.0);
ArrayInitialize(buff3,0.0);
ArrayInitialize(buff4,0.0);
//--- на каждом тике копируем цены в буфера
for(int i=0;i<copied;i++)
{
}

```

```

//--- вычислим соответствующий индекс для буферов
int buffer_index=total-copied+i;
//--- записываем цены в буфера
buff1[buffer_index]=rates[i].open;
buff2[buffer_index]=rates[i].high;
buff3[buffer_index]=rates[i].low;
buff4[buffer_index]=rates[i].close;
//--- зададим цвет свечи
int color_index=i%cncl_colors;
col_buffer[buffer_index]=color_index;
}

return(true);
}

//+-----+
//| Возвращает случайным образом символ из Market Watch |
//+-----+
string GetRandomSymbolName()
{
//--- количество символов, показываемых в окне "Обзор рынка"
int symbols=SymbolsTotal(true);
//--- позиция символа в списке - случайное число от 0 до symbols
int number=MathRand()%symbols;
//--- вернем имя символа по указанной позиции
return SymbolName(number,true);
}

//+-----+
//| Изменяет цвет отрезков свечей |
//+-----+
void ChangeColors(color &cols[],int plot_colors)
{
//--- количество цветов
int size=ArraySize(cols);
//---

string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- для каждого цветового индекса зададим новый цвет случайным образом
for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
{
//--- получим случайное число
int number=MathRand();
//--- получим индекс в массиве col[] как остаток от целочисленного деления
int i=number%size;
//--- установим цвет для каждого индекса как свойство PLOT_LINE_COLOR
PlotIndexSetInteger(0, // номер графического стиля
PLOT_LINE_COLOR, // идентификатор свойства
plot_color_ind, // индекс цвета, куда запишем цвет
cols[i]); // новый цвет

//--- запишем цвета
comm=comm+StringFormat("CandleColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString(cols[i]));
}
}

```

```
    ChartSetString(0,CHART_COMMENT,comm);
}
//---
}
//+-----+
//| Изменяет внешний вид свечей |
//+-----+
void ChangeLineAppearance()
{
//--- строка для формирования информации о свойствах свечей
string comm="";
//--- запишем имя символа
comm="\r\n"+symbol+comm;
//--- выведем информацию на график через комментарий
Comment(comm);
}
```

## Связь между свойствами индикатора и соответствующими функциями

Каждый пользовательский индикатор обладает множеством [свойств](#), часть из которых является обязательной и всегда располагается в самом начале описания. Это свойства:

- указание окна для отображения индикатора - `indicator_separate_window` или `indicator_chart_window`;
- количество индикаторных буферов - `indicator_buffers`;
- количество графических построений, которые выводит индикатора - `indicator_plots`.

Но есть и другая часть свойств, которые можно задавать как через директивы [препроцессора](#), так и через функции для создания пользовательского индикатора. В таблице перечислены эти свойства и соответствующие им функции.

Директивы для свойства подокна индикатора	Функции типа <code>IndicatorSet...()</code>	Описание устанавливаемого свойства подокна
<code>indicator_height</code>	<a href="#"><code>IndicatorSetInteger</code></a> <a href="#"><code>(INDICATOR_INDICATOR_HEIGHT, nHeight)</code></a>	Значение фиксированной высоты подокна
<code>indicator_minimum</code>	<a href="#"><code>IndicatorSetDouble</code></a> <a href="#"><code>(INDICATOR_MINIMUM, d.MaxValue)</code></a>	Минимальное значение вертикальной оси
<code>indicator_maximum</code>	<a href="#"><code>IndicatorSetDouble</code></a> <a href="#"><code>(INDICATOR_MAXIMUM, d.MinValue)</code></a>	Максимальное значение вертикальной оси
<code>indicator_levelN</code>	<a href="#"><code>IndicatorSetDouble</code></a> <a href="#"><code>(INDICATOR_LEVELVALUE, N-1, nLevelValue)</code></a>	Значение уровня номер <code>N</code> на вертикальной оси
нет директивы препроцессора	<a href="#"><code>IndicatorSetString</code></a> <a href="#"><code>(INDICATOR_LEVELTEXT, N-1, sLevelName)</code></a>	Имя отображаемого уровня
<code>indicator_levelcolor</code>	<a href="#"><code>IndicatorSetInteger</code></a> <a href="#"><code>(INDICATOR_LEVELCOLOR, N-1, nLevelColor)</code></a>	Цвет для отображения уровня номер <code>N</code>
<code>indicator_levelwidth</code>	<a href="#"><code>IndicatorSetInteger</code></a> <a href="#"><code>(INDICATOR_LEVELWIDTH, N-1, nLevelWidth)</code></a>	Толщина линии для отображения уровня номер <code>N</code>
<code>indicator_levelstyle</code>	<a href="#"><code>IndicatorSetInteger</code></a> <a href="#"><code>(INDICATOR_LEVELSTYLE, N-1, nLevelStyle)</code></a>	Стиль линии для отображения уровня номер <code>N</code>
Директивы для свойства графических построений	Функции типа <code>PlotIndexSet...()</code>	Описание устанавливаемого

		свойства для графического построения
indicator_labelN	<a href="#">PlotIndexSetString(N-1, PLOT_LABEL, sLabel)</a>	Краткое наименование для графического построения номер N. Показывается в окне DataWindow и во всплывающей подсказке при наведении курсора на построение
indicator_colorN	<a href="#">PlotIndexSetInteger(N-1, PLOT_LINE_COLOR, nColor)</a>	Цвет линии для графического построения номер N
indicator_styleN	<a href="#">PlotIndexSetInteger(N-1, PLOT_LINE_STYLE, nType)</a>	Стиль линии для графического построения номер N
indicator_typeN	<a href="#">PlotIndexSetInteger(N-1, PLOT_DRAW_TYPE, nType)</a>	Тип линии для графического построения номер N
indicator_widthN	<a href="#">PlotIndexSetInteger(N-1, PLOT_LINE_WIDTH, nWidth)</a>	Толщина линии для графического построения номер N
Общие свойства индикатора	Функции типа <a href="#">IndicatorSet...()</a>	Описание
нет директивы препроцессора	<a href="#">IndicatorSetString(INDICATOR_SHORTNAME, sShortName)</a>	Устанавливает удобное короткое имя индикатора, которое будет отображаться в списке индикаторов (вызывается в терминале сочетанием <b>Ctrl+I</b> ).
нет директивы препроцессора	<a href="#">IndicatorSetInteger(INDICATOR_DIGITS, nDigits)</a>	Устанавливает требуемую точность для отображения значений индикатора - количество знаков после десятичной точки
нет директивы препроцессора	<a href="#">IndicatorSetInteger(INDICATOR_LEVELS, nLevels)</a>	Задает количество уровней на окне индикатора
indicator_applied_price	Функции нет, свойство устанавливается только директивой препроцессора.	Тип цены по умолчанию, используемый для расчета значений индикатора. Указывается при необходимости только при использовании функции OnCalculate() первого вида. Значение свойства можно также задавать из диалога свойств индикатора на

		вкладке "Параметры" - <a href="#">"Применить к".</a>
--	--	--

Необходимо отметить, что нумерация уровней и графических построений в терминах препроцессора начинается с единицы, в то время как нумерация тех же самых свойств при использовании функций начинается с нуля, то есть нужно указывать значение на 1 меньше, чем мы указали бы с использованием #property.

Есть несколько директив, для которых не существуют соответствующие функции:

Директива	Описание
indicator_chart_window	Индикатор показывается в главном окне
indicator_separate_window	Индикатор показывается в отдельном подокне
indicator_buffers	Указывает количество требуемых индикаторных буферов
indicator_plots	Указывает количество <a href="#">графических построений</a> в индикаторе

**Смотри также**

[Свойства пользовательских индикаторов](#)

## SetIndexBuffer

Связывает указанный индикаторный буфер с одномерным динамическим массивом типа [double](#).

```
bool SetIndexBuffer(
    int           index,          // индекс буфера
    double        buffer[],       // массив
    ENUM_INDEXBUFFER_TYPE data_type // что будем хранить
);
```

### Параметры

*index*

[in] Номер индикаторного буфера. Нумерация начинается с 0. Номер должен быть меньше значения, объявленного в [#property indicator\\_buffers](#).

*buffer[]*

[in] Массив, объявленный в программе пользовательского индикатора.

*data\_type*

[in] Тип данных, хранящихся в индикаторном массиве. По умолчанию [INDICATOR\\_DATA](#) (значения рассчитанного индикатора). Может также принимать значение [INDICATOR\\_COLOR\\_INDEX](#), тогда данный буфер предназначен для хранения индексов цветов для предыдущего индикаторного буфера. Можно задать до 64 [цветов](#) в строке [#property indicator\\_colorN](#). Значение [INDICATOR\\_CALCULATIONS](#) означает, что данный буфер участвует в промежуточных расчетах индикатора и не предназначен для отрисовки.

### Возвращаемое значение

В случае успешного выполнения возвращает [true](#), в противном случае [false](#).

### Примечание

После связывания динамический массив *buffer[]* будет иметь индексацию как в обычных массивах, даже если для связываемого массива будет предварительно установлена индексация как в [таймсерииях](#). Если необходимо изменить порядок доступа к элементам индикаторного массива, необходимо применить функцию [ArraySetAsSeries\(\)](#) после связывания массива функцией *SetIndexBuffer()*. При этом необходимо иметь ввиду, что нельзя изменять размер для динамических массивов, назначенных в качестве индикаторных буферов функцией *SetIndexBuffer()*. Для индикаторных буферов все операции по изменению размера производит исполняющая подсистема терминала.

### Пример:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//---- plot MA
#property indicator_label1 "MA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
```

```

//--- input parameters
input bool           AsSeries=true;
input int            period=15;
input ENUM_MA_METHOD smootMode=MODE_EMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
input int            shift=0;

//--- indicator buffers
double              MABuffer[];
int                 ma_handle;

//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
    //--- indicator buffers mapping
    if(AsSeries) ArraySetAsSeries(MABuffer,true);
    Print("Индикаторный буфер является таймсерией = ",ArrayGetAsSeries(MABuffer));
    SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
    Print("Индикаторный буфер после SetIndexBuffer() является таймсерией = ",
          ArrayGetAsSeries(MABuffer));

    //--- изменим порядок доступа к элементам индикаторного буфера
    ArraySetAsSeries(MABuffer,AsSeries);

    IndicatorSetString(INDICATOR_SHORTNAME,"MA("+period"+")"+AsSeries);
}

//---
ma_handle=iMA(Symbol(),0,period,shift,smootMode,price);
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- скопируем значения скользящей средней в буфер MABuffer
    int copied=CopyBuffer(ma_handle,0,0,rates_total,MABuffer);

    Print("MABuffer[0] = ",MABuffer[0]); // в зависимости от значения AsSeries
                                         // будем получать либо самое старое значение
}

```

```
// либо на текущем незавершенном баре  
//--- return value of prev_calculated for next call  
    return(rates_total);  
}  
//-----+
```

#### Смотри также

[Свойства пользовательских индикаторов](#), [Доступ к таймсериям и индикаторам](#)

## IndicatorSetDouble

Задаёт значение соответствующего свойства индикатора. Свойство индикатора должно быть типа double. Существует 2 варианта функции.

**Вызов с указанием идентификатора свойства.**

```
bool IndicatorSetDouble(
    int     prop_id,           // идентификатор
    double  prop_value        // устанавливаемое значение
);
```

**Вызов с указанием идентификатора и модификатора свойства.**

```
bool IndicatorSetDouble(
    int     prop_id,           // идентификатор
    int     prop_modifier,      // модификатор
    double  prop_value        // устанавливаемое значение
);
```

### Параметры

*prop\_id*

[in] Идентификатор свойства индикатора. Значение может быть одним из значений перечисления [ENUM\\_CUSTOMIND\\_PROPERTY\\_DOUBLE](#).

*prop\_modifier*

[in] Модификатор указанного свойства. Только свойства уровней требуют модификатора. Нумерация уровней идет с 0, то есть для задания свойства второму уровню нужно указать единицу (на 1 меньше, чем при использовании [директивы компилятора](#)).

*prop\_value*

[in] Значение свойства.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

### Примечание

Нумерация свойств (модификаторов) при использовании директивы #property начинается с 1 (единицы), в то время как функция использует нумерацию с 0 (нуля). При неправильном задании номера уровня [отображение индикатора](#) может отличаться от того, которое предполагается.

Например, задать значение первого уровня для индикатора в отдельном подокне можно двумя способами:

- property indicator\_level1 50 - используется 1 для указания номера уровня,
- IndicatorSetDouble(INDICATOR\_LEVELVALUE, 0, 50) - используется 0 для указания первого уровня.

**Пример:** индикатор-“перевертыш”, меняющий максимальное и минимальное значения окна индикатора, а также значения уровней, на которых расположены горизонтальные линии.



```
#property indicator_separate_window
//--- установим максимальное и минимальное значения для окна индикатора
#property indicator_minimum 0
#property indicator_maximum 100
//--- зададим показ трех горизонтальных уровней в отдельном окне индикатора
#property indicator_level1 25
#property indicator_level2 50
#property indicator_level3 75
//--- установим толщину горизонтальных уровней
#property indicator_levelwidth 1
//--- установим стиль горизонтальных уровней
#property indicator_levelstyle STYLE_DOT
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- зададим описания горизонтальных уровней
    IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
    IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
    IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- зададим короткое имя индикатора
    IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetDouble() Demo");
//--- зададим каждому уровню свой цвет
    IndicatorSetInteger(INDICATOR_LEVELCOLOR,0,clrBlue);
    IndicatorSetInteger(INDICATOR_LEVELCOLOR,1,clrGreen);
    IndicatorSetInteger(INDICATOR_LEVELCOLOR,2,clrRed);
```

```

//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int tick_counter=0;
    static double level1=25,level2=50,level3=75;
    static double max=100,min=0, shift=100;
//--- считаем тики
    tick_counter++;
//--- на каждом 10-м тике делаем переворот
    if(tick_counter%10==0)
    {
//--- перевернем знак для значений уровня
        level1=-level1;
        level2=-level2;
        level3=-level3;
//--- перевернем знак для значений максимума и минимума
        max-=shift;
        min-=shift;
//--- перевернем значение сдвига
        shift=-shift;
//--- зададим новые значения уровней
        IndicatorSetDouble(INDICATOR_LEVELVALUE,0,level1);
        IndicatorSetDouble(INDICATOR_LEVELVALUE,1,level2);
        IndicatorSetDouble(INDICATOR_LEVELVALUE,2,level3);
//--- зададим новые значения максимума и минимума для окна индикатора
        Print("Set up max = ",max,", min = ",min);
        IndicatorSetDouble(INDICATOR_MAXIMUM,max);
        IndicatorSetDouble(INDICATOR_MINIMUM,min);
    }
//--- return value of prev_calculated for next call
    return(rates_total);
}

```

**Смотри также**

[Стили индикаторов в примерах](#), [Связь между свойствами индикатора и функциями](#), [Стили рисования](#)

## IndicatorSetInteger

Задает значение соответствующего свойства индикатора. Свойство индикатора должно быть типа int или color. Существует 2 варианта функции.

**Вызов с указанием идентификатора свойства.**

```
bool IndicatorSetInteger(
    int prop_id,           // идентификатор
    int prop_value         // устанавливаемое значение
);
```

**Вызов с указанием идентификатора и модификатора свойства.**

```
bool IndicatorSetInteger(
    int prop_id,           // идентификатор
    int prop_modifier,     // модификатор
    int prop_value         // устанавливаемое значение
);
```

### Параметры

*prop\_id*

[in] Идентификатор свойства индикатора. Значение может быть одним из значений перечисления [ENUM\\_CUSTOMIND\\_PROPERTY\\_INTEGER](#).

*prop\_modifier*

[in] Модификатор указанного свойства. Только свойства уровней требуют модификатора.

*prop\_value*

[in] Значение свойства.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

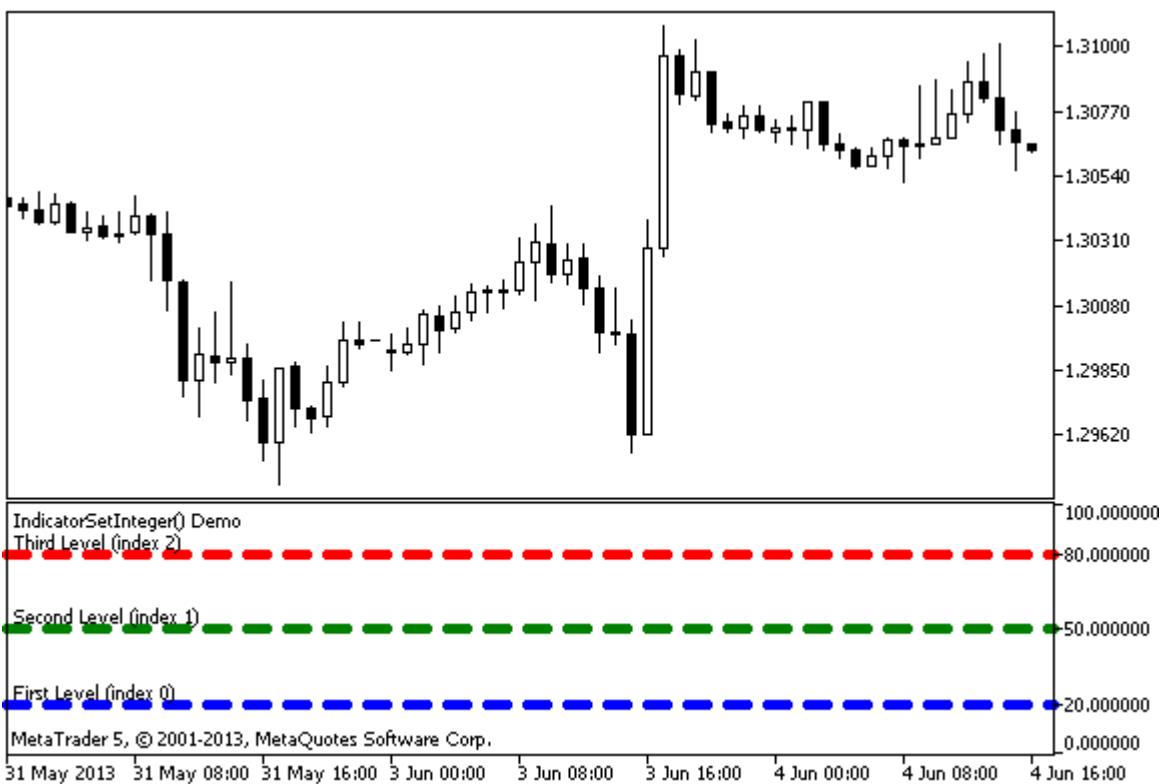
### Примечание

Нумерация свойств (модификаторов) при использовании директивы #property начинается с 1 (единицы), в то время как функция использует нумерацию с 0 (нуля). При неправильном задании номера уровня [отображение индикатора](#) может отличаться от того, которое предполагается.

Например, чтобы задать толщину линии первого горизонтального уровня используйте нулевой индекс:

- `IndicatorSetInteger(INDICATOR_LEVELWIDTH, 0, 5)` - используется индекс 0 для задания толщины линии первого уровня.

**Пример:** индикатор, задающий цвет, стиль и толщину горизонтальных уровней индикатора.



```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- зададим показ трех горизонтальных уровней в отдельном окне индикатора
#property indicator_level1 20
#property indicator_level2 50
#property indicator_level3 80
//--- установим толщину горизонтальных уровней
#property indicator_levelwidth 5
//--- установим цвет горизонтальных уровней
#property indicator_levelcolor clrAliceBlue
//--- установим стиль горизонтальных уровней
#property indicator_levelstyle STYLE_DOT
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- зададим описания горизонтальных уровней
    IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
    IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
    IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- зададим короткое имя индикатора
    IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetInteger() Demo");
    return(INIT_SUCCEEDED);
}
//+-----+
```

```

//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    static int tick_counter=0;
    //--- считаем тики
    tick_counter++;
    //--- и устанавливаем цвета горизонтальных уровней в зависимости от счетчика тиков
    ChangeLevelColor(0,tick_counter,3,6,10); // три последних параметра переключают цвет
    ChangeLevelColor(1,tick_counter,3,6,8);
    ChangeLevelColor(2,tick_counter,4,7,9);
    //--- поменяем стили горизонтальных уровней
    ChangeLevelStyle(0,tick_counter);
    ChangeLevelStyle(1,tick_counter+5);
    ChangeLevelStyle(2,tick_counter+15);
    //--- получим толщину линии как остаток целочисленного деления числа тиков на 5
    int width=tick_counter%5;
    //--- пройдем по всем горизонтальным уровням и выставим
    for(int l=0;l<3;l++)
        IndicatorSetInteger(INDICATOR_LEVELWIDTH,l,width+1);
    //--- return value of prev_calculated for next call
    return(rates_total);
}
//+-----+
//| Установим цвет горизонтальной линии в отдельном окне индикатора | 
//+-----+
void ChangeLevelColor(int level,           // номер горизонтальной линии
                      int tick_number, // делимое, число для получения остатка деления
                      int f_trigger,  // первый делитель переключения цвета
                      int s_trigger,  // второй делитель переключения цвета
                      int t_trigger) // третий делитель переключения цвета
{
    static color colors[3]={clrRed,clrBlue,clrGreen};
    //--- индекс цвета из массива colors[]
    int index=-1;
    //--- вычислим номер цвета из массива colors[] для раскрашивания горизонтальной линии
    if(tick_number%f_trigger==0)
        index=0; // если число tick_number делится без остатка на f_trigger
    if(tick_number%s_trigger==0)
        index=1; // если число tick_number делится без остатка на s_trigger
}

```

```

if(tick_number%t_trigger==0)
    index=2; // если число tick_number делится без остатка на t_trigger
//--- если цвет определен, установим его
if(index!=-1)
    IndicatorSetInteger(INDICATOR_LEVELCOLOR,level,colors[index]);
//---
}

//+-----+
//| Установим стиль горизонтальной линии в отдельном окне индикатора |
//+-----+
void ChangeLevelStyle(int level, // номер горизонтальной линии
                      int tick_number// число для получения остатка от деления
                     )
{
//--- массив для хранения стилей
static ENUM_LINE_STYLE styles[5]=
{STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT};
//--- индекс стиля из массива styles[]
int index=-1;
//--- вычислим номер из массива styles[] для установки стиля горизонтальной линии
if(tick_number%50==0)
    index=5; // если tick_number делится без остатка на 50, то стиль STYLE_DASHDOT
if(tick_number%40==0)
    index=4; // ... стиль STYLE_DASHDOT
if(tick_number%30==0)
    index=3; // ... STYLE_DOT
if(tick_number%20==0)
    index=2; // ... STYLE_DASH
if(tick_number%10==0)
    index=1; // ... STYLE_SOLID
//--- если стиль определен, установим его
if(index!=-1)
    IndicatorSetInteger(INDICATOR_LEVELSTYLE,level,styles[index]);
}

```

#### Смотри также

[Свойства пользовательских индикаторов](#), [Свойства программ \(#property\)](#), [Стили рисования](#)

## IndicatorSetString

Задает значение соответствующего свойства индикатора. Свойство индикатора должно быть типа `string`. Существует 2 варианта функции.

**Вызов с указанием идентификатора свойства.**

```
bool IndicatorSetString(
    int     prop_id,           // идентификатор
    string  prop_value        // устанавливаемое значение
);
```

**Вызов с указанием идентификатора и модификатора свойства.**

```
bool IndicatorSetString(
    int     prop_id,           // идентификатор
    int     prop_modifier,      // модификатор
    string  prop_value        // устанавливаемое значение
);
```

### Параметры

`prop_id`

[in] Идентификатор свойства индикатора. Значение может быть одним из значений перечисления [ENUM\\_CUSTOMIND\\_PROPERTY\\_STRING](#).

`prop_modifier`

[in] Модификатор указанного свойства. Только свойства уровней требуют модификатора.

`prop_value`

[in] Значение свойства.

### Возвращаемое значение

В случае успешного выполнения возвращает `true`, в противном случае `false`.

### Примечание

Нумерация свойств (модификаторов) при использовании директивы `#property` начинается с 1 (единицы), в то время как функция использует нумерацию с 0 (нуля). При неправильном задании номера уровня [отображение индикатора](#) может отличаться от того, которое предполагается.

Например, чтобы задать описание первого горизонтального уровня, используйте нулевой индекс:

- `IndicatorSetString(INDICATOR_LEVELTEXT, 0, "First Level")` - используется индекс 0 для задания текстового описания первого уровня.

**Пример:** индикатор, устанавливающий подписи к горизонтальным уровням индикатора.



```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- зададим показ трех горизонтальных уровней в отдельном окне индикатора
#property indicator_level1 30
#property indicator_level2 50
#property indicator_level3 70
//--- установим цвет горизонтальных уровней
#property indicator_levelcolor clrRed
//--- установим стиль горизонтальных уровней
#property indicator_levelstyle STYLE_SOLID
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- зададим описания горизонтальных уровней
    IndicatorSetString(INDEX_LEVELTEXT,0,"First Level (index 0)");
    IndicatorSetString(INDEX_LEVELTEXT,1,"Second Level (index 1)");
    IndicatorSetString(INDEX_LEVELTEXT,2,"Third Level (index 2)");
//--- зададим короткое имя индикатора
    IndicatorSetString(INDEX_SHORTNAME,"IndicatorSetString() Demo");
//---
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function               |
//+-----+
```

```
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//---

//--- return value of prev_calculated for next call
    return(rates_total);
}
```

#### Смотри также

[Свойства пользовательских индикаторов](#), [Свойства программ \(#property\)](#)

## PlotIndexSetDouble

Задает значение соответствующего свойства соответствующей линии индикатора. Свойство индикатора должно быть типа double.

```
bool PlotIndexSetDouble(
    int    plot_index,      // индекс графического стиля
    int    prop_id,         // идентификатор свойства
    double prop_value       // устанавливаемое значение
);
```

### Параметры

*plot\_index*

[in] Индекс [графического построения](#)

*prop\_id*

[in] Идентификатор свойства индикатора. Значение может быть одним из значений перечисления [ENUM\\_PLOT\\_PROPERTY\\_DOUBLE](#).

*prop\_value*

[in] Значение свойства.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## PlotIndexSetInteger

Задает значение соответствующего свойства соответствующей линии индикатора. Свойство индикатора должно быть типа int, char, bool или color. Существует 2 варианта функции.

**Вызов с указанием идентификатора свойства.**

```
bool PlotIndexSetInteger(
    int plot_index,           // индекс графического стиля
    int prop_id,              // идентификатор свойства
    int prop_value            // устанавливаемое значение
);
```

**Вызов с указанием идентификатора и модификатора свойства.**

```
bool PlotIndexSetInteger(
    int plot_index,           // индекс графического стиля
    int prop_id,              // идентификатор свойства
    int prop_modifier,         // модификатор свойства
    int prop_value            // устанавливаемое значение
);
```

### Параметры

*plot\_index*

[in] Индекс [графического построения](#)

*prop\_id*

[in] Идентификатор свойства индикатора. Значение может быть одним из значений перечисления [ENUM\\_PLOT\\_PROPERTY\\_INTEGER](#).

*prop\_modifier*

[in] Модификатор указанного свойства. Только свойства индексов цветов требуют модификатора.

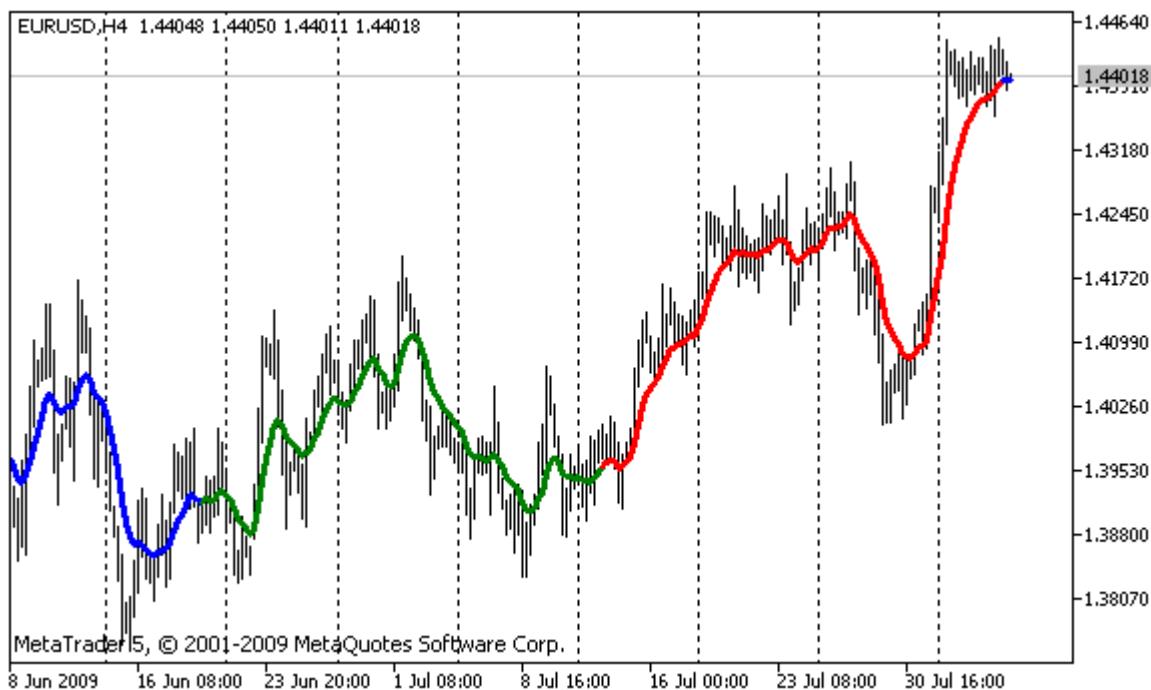
*prop\_value*

[in] Значение свойства.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

**Пример:** индикатор, рисующий трехцветную линию. Цветовая схема меняется через каждые 5 тиков.



```

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots 1
//---- plot ColorLine
#property indicator_label1 "ColorLine"
#property indicator_type1 DRAW_COLOR_LINE
#property indicator_color1 clrRed,clrGreen,clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 3
//--- indicator buffers
double ColorLineBuffer[];
double ColorBuffer[];
int MA_handle;
//+-----+
//| Custom indicator initialization function |
//+-----+
void OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,ColorLineBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,ColorBuffer,INDICATOR_COLOR_INDEX);
//--- get MA handle
    MA_handle=iMA(Symbol(),0,10,0,MODE_EMA,PRICE_CLOSE);
//---
}
//+-----+
//| get color index |
//+-----+

```

```

int getIndexOfColor(int i)
{
    int j=i%300;
    if(j<100) return(0); // first index
    if(j<200) return(1); // second index
    return(2); // third index
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//---
    static int ticks=0,modified=0;
    int limit;
//--- first calculation or number of bars was changed
    if(prev_calculated==0)
    {
        //--- copy values of MA into indicator buffer ColorLineBuffer
        int copied=CopyBuffer(MA_handle,0,0,rates_total,ColorLineBuffer);
        if(copied<=0) return(0); // copying failed - throw away
        //--- now set line color for every bar
        for(int i=0;i<rates_total;i++)
            ColorBuffer[i]=getIndexOfColor(i);
    }
    else
    {
        //--- copy values of MA into indicator buffer ColorLineBuffer
        int copied=CopyBuffer(MA_handle,0,0,rates_total,ColorLineBuffer);
        if(copied<=0) return(0);

        ticks++; // ticks counting
        if(ticks>=5)//it's time to change color scheme
        {
            ticks=0; // reset counter
            modified++; // counter of color changes
            if(modified>=3)modified=0;// reset counter
            ResetLastError();
            switch(modified)
            {

```

```
case 0:// first color scheme
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrRed);
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrBlue);
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrGreen);
    Print("Color scheme "+modified);
    break;
case 1:// second color scheme
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrYellow);
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrPink);
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrLightSlateGray);
    Print("Color scheme "+modified);
    break;
default:// third color scheme
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrLightGoldenrod);
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrOrchid);
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrLimeGreen);
    Print("Color scheme "+modified);
}
}
else
{
//--- set start position
limit=prev_calculated-1;
//--- now we set line color for every bar
for(int i=limit;i<rates_total;i++)
    ColorBuffer[i]=getIndexOfColor(i);
}
}
//--- return value of prev_calculated for next call
return(rates_total);
}
//-----+
```

## PlotIndexSetString

Задает значение соответствующего свойства соответствующей линии индикатора. Свойство индикатора должно быть типа `string`.

```
bool PlotIndexSetString(  
    int    plot_index,      // индекс графического стиля  
    int    prop_id,        // идентификатор свойства  
    string prop_value      // устанавливаемое значение  
) ;
```

### Параметры

*plot\_index*

[in] Индекс [графического построения](#)

*prop\_id*

[in] Идентификатор свойства индикатора. Значение может быть одним из значений перечисления [ENUM\\_PLOT\\_PROPERTY\\_STRING](#).

*prop\_value*

[in] Значение свойства.

### Возвращаемое значение

В случае успешного выполнения возвращает `true`, в противном случае `false`.

## PlotIndexGetInteger

Возвращает значение соответствующего свойства соответствующей линии индикатора. Свойство должно быть типов int, color, bool или char. Существует 2 варианта функции.

**Вызов с указанием идентификатора свойства.**

```
int PlotIndexGetInteger(
    int plot_index,           // индекс графического стиля
    int prop_id,              // идентификатор свойства
)
;
```

**Вызов с указанием идентификатора и модификатора свойства.**

```
int PlotIndexGetInteger(
    int plot_index,           // индекс графического стиля
    int prop_id,              // идентификатор свойства
    int prop_modifier         // модификатор свойства
)
;
```

### Параметры

*plot\_index*

[in] Индекс [графического построения](#)

*prop\_id*

[in] Идентификатор свойства индикатора. Значение может быть одним из значений перечисления [ENUM\\_PLOT\\_PROPERTY\\_INTEGER](#).

*prop\_modifier*

[in] Модификатор указанного свойства. Только свойства индексов цветов требуют модификатора.

### Примечание

Функция предназначена для извлечения настроек рисования соответствующей линии индикатора. Функция работает в паре с функцией [PlotIndexSetInteger](#) для копирования свойств рисования из одной линии в другую.

**Пример:** индикатор, окрашивающий свечи в цвет, зависящий от дня недели. Цвета для каждого дня задаются программным путем.



```

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots 1
//---- plot ColorCandles
#property indicator_label1 "ColorCandles"
#property indicator_type1 DRAW_COLOR_CANDLES
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- indicator buffers
double OpenBuffer[];
double HighBuffer[];
double LowBuffer[];
double CloseBuffer[];
double ColorCandlesColors[];
color ColorOfDay[6]={CLR_NONE,clrMediumSlateBlue,
                    clrDarkGoldenrod,clrForestGreen,clrBlueViolet,clrRed};
//-----+
//| Custom indicator initialization function |
//-----+
void OnInit()
{
//--- indicator buffers mapping
    SetIndexBuffer(0,OpenBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,HighBuffer,INDICATOR_DATA);
    SetIndexBuffer(2,LowBuffer,INDICATOR_DATA);
    SetIndexBuffer(3,CloseBuffer,INDICATOR_DATA);
    SetIndexBuffer(4,ColorCandlesColors,INDICATOR_COLOR_INDEX);
//--- set number of colors in color buffer

```

```
PlotIndexSetInteger(0,PLOT_COLOR_INDEXES,6);
//--- set colors for color buffer
for(int i=1;i<6;i++)
    PlotIndexSetInteger(0,PLOT_LINE_COLOR,i,ColorOfDay[i]);
//--- set accuracy
IndicatorSetInteger(INDEX_DIGITS,_Digits);
printf("We have %u colors of days",PlotIndexGetInteger(0,PLOT_COLOR_INDEXES));
//---

}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//---
int i;
MqlDateTime t;
//---
if(prev_calculated==0) i=0;
else i=prev_calculated-1;
//---
while(i<rates_total)
{
    OpenBuffer[i]=open[i];
    HighBuffer[i]=high[i];
    LowBuffer[i]=low[i];
    CloseBuffer[i]=close[i];
//--- set color for every candle
TimeToStruct(time[i],t);
ColorCandlesColors[i]=t.day_of_week;
//---
i++;
}
//--- return value of prev_calculated for next call
return(rates_total);
}
//+-----+
```

## Графические объекты

Группа функций, предназначенных для работы с графическими объектами, относящимися к любому указанному графику.

Функции, задающие свойства графических объектов, а также операции создания [ObjectCreate\(\)](#) и перемещения [ObjectMove\(\)](#) объектов на графике фактически служат для отправки команды графику. При успешном выполнении этих функций команда попадает в общую очередь событий графика. Визуальное изменение свойств графических объектов производится в процессе обработки очереди событий данного графика.

По этой причине не следует ожидать немедленного визуального обновления графических объектов после вызова данных функций. В общем случае обновление графических объектов на чарте производится терминалом автоматически по событиям изменения - поступление новой котировки, изменения размера окна графика и т.д. Для принудительного обновления графических объектов используйте команду на перерисовку графика [ChartRedraw\(\)](#).

Функция	Действие
<a href="#">ObjectCreate</a>	Создает объект заданного типа на указанном графике
<a href="#">ObjectName</a>	Возвращает имя объекта соответствующего типа в указанном графике (указанном подокне графика)
<a href="#">ObjectDelete</a>	Удаляет объект с указанным именем с указанного графика (с указанного подокна графика)
<a href="#">ObjectsDeleteAll</a>	Удаляет все объекты указанного типа с указанного графика (с указанного подокна графика)
<a href="#">ObjectFind</a>	Ищет по имени объект с указанным идентификатором
<a href="#">ObjectGetTimeByValue</a>	Возвращает значение времени для указанного значения цены объекта
<a href="#">ObjectGetValueByTime</a>	Возвращает ценовое значение объекта для указанного времени
<a href="#">ObjectMove</a>	Изменяет координаты указанной точки привязки объекта
<a href="#">ObjectsTotal</a>	Возвращает количество объектов указанного типа в указанном графике (указанном подокне графика)
<a href="#">ObjectGetDouble</a>	Возвращает значение типа double соответствующего свойства объекта
<a href="#">ObjectGetInteger</a>	Возвращает целочисленное значение соответствующего свойства объекта

<a href="#">ObjectGetString</a>	Возвращает значение типа string соответствующего свойства объекта
<a href="#">ObjectSetDouble</a>	Устанавливает значение соответствующего свойства объекта
<a href="#">ObjectSetInteger</a>	Устанавливает значение соответствующего свойства объекта
<a href="#">ObjectSetString</a>	Устанавливает значение соответствующего свойства объекта
<a href="#">TextSetFont</a>	Устанавливает шрифт для вывода текста методами рисования (по умолчанию используется шрифт Arial 20)
<a href="#">TextOut</a>	Выводит текст в пользовательский массив (буфер), предназначенный для создания графического <a href="#">ресурса</a>
<a href="#">TextGetSize</a>	Возвращает ширину и высоту строки при текущих <a href="#">настройках шрифта</a>

Каждый графический объект должен иметь имя, уникальное в пределах одного [графика](#), включая его подокна. Изменение имени графического объекта формирует два события: первое - это событие удаления объекта со старым именем, и второе - событие создания графического объекта с новым именем.

После создания объекта или модификации [свойств объекта](#) рекомендуется вызывать функцию [ChartRedraw\(\)](#), которая отдает терминалу команду на принудительную отрисовку графика (и всех [видимых](#) на нем объектов).

## ObjectCreate

Создает объект с указанным именем, типом и начальными координатами в указанном подокне графика. При создании можно указать до 30 координат.

```
bool ObjectCreate(
    long      chart_id,           // идентификатор графика
    string    name,              // имя объекта
    ENUM_OBJECT type,            // тип объекта
    int       sub_window,        // индекс окна
    datetime  time1,             // время первой точки привязки
    double    price1,            // цена первой точки привязки
    ...
    datetime  timeN=0,           // время N-ой точки привязки
    double    priceN=0,          // цена N-ой точки привязки
    ...
    datetime  time30=0,           // время 30-й точки привязки
    double    price30=0          // цена 30-точки привязки
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя объекта. Имя должно быть уникальным в пределах одного графика, включая его подокна.

*type*

[in] Тип объекта. Значение может быть одним из значений перечисления [ENUM\\_OBJECT](#).

*sub\_window*

[in] Номер подокна графика. 0 означает главное окно графика. Указанное подокно должно существовать, в противном случае функция возвращает false.

*time1*

[in] Временная координата первой привязки.

*price1*

[in] Ценовая координата первой точки привязки.

*timeN=0*

[in] Временная координата N-ой точки привязки.

*priceN=0*

[in] Ценовая координата N-ой точки привязки.

*time30=0*

[in] Временная координата тридцатой точки привязки.

*price30=0*

[in] Ценовая координата тридцатой точки привязки.

### Возвращаемое значение

Возвращает true при успешной постановке команды в очередь указанного графика, иначе false. Если объект был уже создан ранее, то производится попытка изменить его координаты.

### Примечание

При вызове ObjectCreate() всегда используется асинхронный вызов, поэтому функция возвращает только результат постановки команды в очередь графика. В этом случае true означает только то, что команда успешно поставлена в очередь, сам результат её выполнения неизвестен.

Для проверки результата выполнения можно использовать функцию [ObjectFind\(\)](#) или любые функции, запрашивающие свойства объекта, например вида ObjectGetXXX. Но при этом следует иметь в виду, что такие функции ставятся в конец очереди команд графика и дожидаются результата выполнения (так как являются синхронными вызовами), то есть могут быть затратными по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

Имя графического объекта не должно превышать 63 символа.

Нумерация подокон графика (если на графике есть подокна с индикаторами) начинается с 1. Главное окно графика есть всегда и имеет индекс 0.

Большое количество точек привязки (до 30-ти) предусмотрено для будущего использования. В то же время ограничение только 30-тью возможными точками привязки для графических объектов обусловлено тем, что при вызове функции количество параметров не должно превышать 64.

При переименовании графического объекта одновременно формируются два события, которые можно обработать в эксперте или индикаторе функцией [OnChartEvent\(\)](#):

- событие удаления объекта со старым именем;
- событие создания графического объекта с новым именем.

Для создания каждого из [типов объектов](#) требуется задать определенное количество точек привязки:

Идентификатор	Описание	Точки привязки
<a href="#">OBJ_VLINE</a>	Вертикальная линия	Одна точка привязки. Фактически используется только координата по оси времени.
<a href="#">OBJ_HLINE</a>	Горизонтальная линия	Одна точка привязки. Фактически используется только координата по оси цены.
<a href="#">OBJ_TREND</a>	Трендовая линия	Две точки привязки.
<a href="#">OBJ TREND BY ANGLE</a>	Трендовая линия по углу	Две точки привязки.
<a href="#">OBJ_CYCLES</a>	Циклические линии	Две точки привязки.

<a href="#">OBJ_ARROWED_LINE</a>	Объект "Линия со стрелкой"	Две точки привязки.
<a href="#">OBJ_CHANNEL</a>	Равноудаленный канал	Три точки привязки.
<a href="#">OBJ_STDDEVCHANNEL</a>	Канал стандартного отклонения	Две точки привязки.
<a href="#">OBJ_REGRESSION</a>	Канал на линейной регрессии	Две точки привязки.
<a href="#">OBJ_PITCHFORK</a>	Вилы Эндрюса	Три точки привязки.
<a href="#">OBJ_GANNLINE</a>	Линия Ганна	Две точки привязки.
<a href="#">OBJ_GANNFAN</a>	Веер Ганна	Две точки привязки.
<a href="#">OBJ_GANNGRID</a>	Сетка Ганна	Две точки привязки.
<a href="#">OBJ_FIBO</a>	Уровни Фибоначчи	Две точки привязки.
<a href="#">OBJ_FIBOTIMES</a>	Временные зоны Фибоначчи	Две точки привязки.
<a href="#">OBJ_FIBOFAN</a>	Веер Фибоначчи	Две точки привязки.
<a href="#">OBJ_FIBOARC</a>	Дуги Фибоначчи	Две точки привязки.
<a href="#">OBJ_FIBOCHANNEL</a>	Канал Фибоначчи	Три точки привязки.
<a href="#">OBJ_EXPANSION</a>	Расширение Фибоначчи	Три точки привязки.
<a href="#">OBJ_ELLIOTWAVE5</a>	5-волновка Эллиота	Пять точек привязки.
<a href="#">OBJ_ELLIOTWAVE3</a>	3-волновка Эллиота	Три точки привязки.
<a href="#">OBJ_RECTANGLE</a>	Прямоугольник	Две точки привязки.
<a href="#">OBJ_TRIANGLE</a>	Треугольник	Три точки привязки.
<a href="#">OBJ_ELLIPSE</a>	Эллипс	Три точки привязки.
<a href="#">OBJ_ARROW_THUMB_UP</a>	Знак "Хорошо" (большой палец вверх)	Одна точка привязки.
<a href="#">OBJ_ARROW_THUMB_DOWN</a>	Знак "Плохо" (большой палец вниз)	Одна точка привязки.
<a href="#">OBJ_ARROW_UP</a>	Знак "Стрелка вверх"	Одна точка привязки.
<a href="#">OBJ_ARROW_DOWN</a>	Знак "Стрелка вниз"	Одна точка привязки.
<a href="#">OBJ_ARROW_STOP</a>	Знак "Стоп"	Одна точка привязки.
<a href="#">OBJ_ARROW_CHECK</a>	Знак "Птичка" (галка)	Одна точка привязки.
<a href="#">OBJ_ARROW_LEFT_PRICE</a>	Левая ценовая метка	Одна точка привязки.
<a href="#">OBJ_ARROW_RIGHT_PRICE</a>	Правая ценовая метка	Одна точка привязки.
<a href="#">OBJ_ARROW_BUY</a>	Знак "Buy"	Одна точка привязки.
<a href="#">OBJ_ARROW_SELL</a>	Знак "Sell"	Одна точка привязки.
<a href="#">OBJ_ARROW</a>	Объект "Стрелка"	Одна точка привязки.

<u>OBJ_TEXT</u>	Объект "Текст"	Одна точка привязки.
<u>OBJ_LABEL</u>	Объект "Текстовая метка"	Положение задается при помощи свойств <a href="#">OBJPROP_XDISTANCE</a> и <a href="#">OBJPROP_YDISTANCE</a> .
<u>OBJ_BUTTON</u>	Объект "Кнопка"	Положение задается при помощи свойств <a href="#">OBJPROP_XDISTANCE</a> и <a href="#">OBJPROP_YDISTANCE</a> .
<u>OBJ_CHART</u>	Объект "График"	Положение задается при помощи свойств <a href="#">OBJPROP_XDISTANCE</a> и <a href="#">OBJPROP_YDISTANCE</a> .
<u>OBJ_BITMAP</u>	Объект "Рисунок"	Одна точка привязки.
<u>OBJ_BITMAP_LABEL</u>	Объект "Графическая метка"	Положение задается при помощи свойств <a href="#">OBJPROP_XDISTANCE</a> и <a href="#">OBJPROP_YDISTANCE</a> .
<u>OBJ_EDIT</u>	Объект "Поле ввода"	Положение задается при помощи свойств <a href="#">OBJPROP_XDISTANCE</a> и <a href="#">OBJPROP_YDISTANCE</a> .
<u>OBJ_EVENT</u>	Объект "Событие", соответствующий событию в экономическом календаре	Одна точка привязки. Фактически используется только координата по оси времени.
<u>OBJ_RECTANGLE_LABEL</u>	Объект "Прямоугольная метка" для создания и оформления пользовательского графического интерфейса.	Положение задается при помощи свойств <a href="#">OBJPROP_XDISTANCE</a> и <a href="#">OBJPROP_YDISTANCE</a> .

## ObjectName

Возвращает имя соответствующего объекта в указанном чарте, указанном подокне указанного чарта, указанного типа.

```
string ObjectName(
    long chart_id,           // идентификатор графика
    int pos,                 // номер в списке объектов
    int sub_window=-1,        // номер окна
    int type=-1              // тип объекта
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*pos*

[in] Порядковый номер объекта согласно указанного фильтра по номеру подокна и типу.

*sub\_window=-1*

[in] Номер подокна графика. 0 означает главное окно графика, -1 означает все подокна графика, включая главное окно.

*type=-1*

[in] Тип объекта. Значение может быть одним из значений перечисления [ENUM\\_OBJECT](#). -1 означает все типы.

### Возвращаемое значение

Имя объекта в случае успеха.

### Примечание

Функция использует синхронный вызов - это означает, что функция дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом, и поэтому данная функция может быть затратной по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

При переименовании графического объекта одновременно формируются два события, которые можно обработать в эксперте или индикаторе функцией [OnChartEvent\(\)](#):

- событие удаления объекта со старым именем;
- событие создания графического объекта с новым именем.

## ObjectDelete

Удаляет объект с указанным именем с указанного графика.

```
bool ObjectDelete(
    long    chart_id,      // идентификатор графика
    string  name          // имя объекта
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя удаляемого объекта.

### Возвращаемое значение

Возвращает true при успешной постановке команды в очередь указанного графика, иначе false.

### Примечание

При вызове ObjectDelete() всегда используется асинхронный вызов, поэтому функция возвращает только результат постановки команды в очередь графика. В этом случае true означает только то, что команда успешно поставлена в очередь, сам результат её выполнения неизвестен.

Для проверки результата выполнения можно использовать функцию [ObjectFind\(\)](#) или любые функции, запрашивающие свойства объекта, например вида ObjectGetXXX. Но при этом следует иметь в виду, что такие функции ставятся в конец очереди команд графика и дожидаются результата выполнения (так как являются синхронными вызовами), то есть могут быть затратными по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

При переименовании графического объекта одновременно формируются два события, которые можно обработать в эксперте или индикаторе функцией [OnChartEvent\(\)](#):

- событие удаления объекта со старым именем;
- событие создания графического объекта с новым именем.

## ObjectsDeleteAll

Удаляет все объекты в указанном графике, указанном подокне указанного графика, указанного типа. Существует два варианта функции:

```
int ObjectsDeleteAll(
    long   chart_id,           // идентификатор графика
    int    sub_window=-1,       // индекс окна
    int    type=-1             // тип объекта для удаления
);
```

Удаляет по префиксу имени в подокне графика все объекты указанного типа.

```
int ObjectsDeleteAll(
    long      chart_id,        // идентификатор графика
    const string prefix,        // префикс имени объекта
    int       sub_window=-1,    // индекс окна
    int       object_type=-1   // тип объекта для удаления
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*prefix*

[in] Префикс, по которому будут удалены все объекты, чьи имена начинаются с данного набора символов. Префикс можно указывать как 'name' или 'name\*' - оба варианта работают одинаково. Если в качестве префикса указана пустая строка, то будут удалены объекты с любым именем.

*sub\_window=-1*

[in] Номер подокна графика. 0 означает главное окно графика, -1 означает все подокна графика, включая главное окно.

*type=-1*

[in] Тип объекта. Значение может быть одним из значений перечисления [ENUM\\_OBJECT](#). -1 означает все типы.

### Возвращаемое значение

Возвращает количество удаленных объектов. Для получения дополнительной информации об ошибке необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция использует синхронный вызов - это означает, что функция дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом, и поэтому данная функция может быть затратной по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

## ObjectFind

Ищет объект с указанным именем на графике с указанным идентификатором.

```
int ObjectFind(
    long    chart_id,      // идентификатор графика
    string  name          // имя объекта
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя искомого объекта.

### Возвращаемое значение

В случае удачи функция возвращает номер подокна (0 означает главное окно графика), в котором находится найденный объект. Если объект не найден, то функция возвращает отрицательное число. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция использует синхронный вызов - это означает, что функция дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом, и поэтому данная функция может быть затратной по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

При переименовании графического объекта одновременно формируются два события, которые можно обработать в эксперте или индикаторе функцией [OnChartEvent\(\)](#):

- событие удаления объекта со старым именем;
- событие создания графического объекта с новым именем.

## ObjectGetTimeByValue

Возвращает значение времени для указанного значения цены указанного объекта.

```
datetime ObjectGetTimeByValue(
    long    chart_id,      // идентификатор графика
    string  name,         // имя объекта
    double  value,        // цена
    int     line_id       // номер линии
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя объекта.

*value*

[in] Значение цены.

*line\_id*

[in] Идентификатор линии.

### Возвращаемое значение

Значение времени для указанного значения цены указанного объекта.

### Примечание

Функция использует синхронный вызов - это означает, что функция дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом, и поэтому данная функция может быть затратной по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

Так как объект в одной координате цены может иметь несколько значений, то необходимо указать номер линии. Эта функция применима только для следующих объектов:

- Трендовая линия (OBJ\_TREND)
- Трендовая линия по углу (OBJ\_TREND\_BY\_ANGLE)
- Линия Ганна (OBJ\_GANNLINE)
- Равноудаленный канал (OBJ\_CHANNEL) - 2 линии
- Канал на линейной регрессии (OBJ\_REGRESSION) - 3 линии
- Канал стандартного отклонения (OBJ\_STDDEVCHANNEL) - 3 линии
- Линия со стрелкой (OBJ\_ARROWED\_LINE)

### Смотри также

[Типы объектов](#)

## ObjectGetValueByTime

Возвращает значение цены для указанного времени указанного объекта.

```
double ObjectGetValueByTime(
    long      chart_id,        // идентификатор графика
    string    name,           // имя объекта
    datetime time,           // время
    int       line_id         // номер линии
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя объекта.

*time*

[in] Значение времени.

*line\_id*

[in] Идентификатор линии.

### Возвращаемое значение

Значение цены для указанного времени указанного объекта.

### Примечание

Функция использует синхронный вызов - это означает, что функция дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом, и поэтому данная функция может быть затратной по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

Так как объект в одной координате цены может иметь несколько значений, то необходимо указать номер линии. Эта функция применима только для следующих объектов:

- Трендовая линия (OBJ\_TREND)
- Трендовая линия по углу (OBJ\_TREND\_BY\_ANGLE)
- Линия Ганна (OBJ\_GANNLINE)
- Равноудаленный канал (OBJ\_CHANNEL) - 2 линии
- Канал на линейной регрессии (OBJ\_REGRESSION) - 3 линии
- Канал стандартного отклонения (OBJ\_STDDEVCHANNEL) - 3 линии
- Линия со стрелкой (OBJ\_ARROWED\_LINE)

### Смотри также

[Типы объектов](#)

## ObjectMove

Изменяет координаты указанной точки привязки объекта.

```
bool ObjectMove(
    long      chart_id,           // идентификатор графика
    string    name,              // имя объекта
    int       point_index,        // номер привязки
    datetime  time,              // время
    double    price              // цена
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя объекта.

*point\_index*

[in] Номер точки привязки. Количество точек привязки зависит от типа объекта.

*time*

[in] Временная координата указанной точки привязки.

*price*

[in] Ценовая координата указанной точки привязки.

### Возвращаемое значение

Возвращает true при успешной постановке команды в очередь указанного графика, иначе false.

### Примечание

При вызове ObjectMove() всегда используется асинхронный вызов, поэтому функция возвращает только результат постановки команды в очередь графика. В этом случае true означает только то, что команда успешно поставлена в очередь, сам результат её выполнения неизвестен.

Для проверки результата выполнения можно использовать функцию, запрашивающую свойства объекта, например вида ObjectGetXXX. Но при этом следует иметь в виду, что такие функции ставятся в конец очереди команд графика и дожидаются результата выполнения (так как являются синхронными вызовами), то есть могут быть затратными по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

## ObjectsTotal

Возвращает количество объектов в указанном чарте, указанном подокне указанного чарта, указанного типа.

```
int ObjectsTotal(
    long chart_id,           // идентификатор графика
    int sub_window=-1,       // индекс окна
    int type=-1              // тип объекта
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*nwin=-1*

[in] Номер подокна графика. 0 означает главное окно графика, -1 означает все подокна графика, включая главное окно.

*type=-1*

[in] Тип объекта. Значение может быть одним из значений перечисления [ENUM\\_OBJECT](#). -1 означает все типы.

### Возвращаемое значение

Количество объектов.

### Примечание

Функция использует синхронный вызов - это означает, что функция дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом, и поэтому данная функция может быть затратной по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

## ObjectSetDouble

Задает значение соответствующего свойства объекта. Свойство объекта должно быть типа [double](#). Существует 2 варианта функции.

### Установка значения свойства, не имеющего модификатора

```
bool ObjectSetDouble(
    long             chart_id,           // идентификатор графика
    string           name,              // имя
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id, // свойство
    double           prop_value        // значение
);
```

### Установка значения свойства с указанием модификатора

```
bool ObjectSetDouble(
    long             chart_id,           // идентификатор графика
    string           name,              // имя
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id, // свойство
    int              prop_modifier,     // модификатор
    double           prop_value        // значение
);
```

#### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя объекта.

*prop\_id*

[in] Идентификатор свойства объекта. Значение может быть одним из значений перечисления [ENUM\\_OBJECT\\_PROPERTY\\_DOUBLE](#).

*prop\_modifier*

[in] Модификатор указанного свойства. Означает номер уровня в [инструментах Фибоначчи](#) и в графическом объекте Вилы Эндрюса. Нумерация уровней начинается с нуля.

*prop\_value*

[in] Значение свойства.

#### Возвращаемое значение

Возвращает true только в том случае, если команда на изменение свойств графического объекта успешно отправлена графику, иначе возвращает false. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

#### Примечание

Функция использует асинхронный вызов - это означает, что функция не дожидается выполнения команды, успешно поставленной в очередь указанного графика, а сразу же возвращает управление.

Для проверки результата выполнения на чужом графике можно использовать функцию, запрашивающую указанное свойство объекта. Но при этом следует иметь в виду, что такие функции ставятся в конец очереди команд чужого графика и дожидаются результата выполнения, то есть могут быть затратными по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

### Пример создания Фибо-объекта и добавления нового уровня в нем

```
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
    //--- вспомогательные массивы
    double high[],low[],price1,price2;
    datetime time[],time1,time2;
    //--- скопируем цены открытия - 100 последних баров хватит
    int copied=CopyHigh(Symbol(),0,0,100,high);
    if(copied<=0)
    {
        Print("Не удалось скопировать значения ценовой серии High");
        return;
    }
    //--- скопируем цены закрытия - 100 последних баров хватит
    copied=CopyLow(Symbol(),0,0,100,low);
    if(copied<=0)
    {
        Print("Не удалось скопировать значения ценовой серии Low");
        return;
    }
    //--- скопируем время открытия для 100 последних баров
    copied=CopyTime(Symbol(),0,0,100,time);
    if(copied<=0)
    {
        Print("Не удалось скопировать значения ценовой серии Time");
        return;
    }
    //--- организуем доступ к скопированным данным как к таймсериям - задом наперед
    ArraySetAsSeries(high,true);
    ArraySetAsSeries(low,true);
    ArraySetAsSeries(time,true);

    //--- координаты первой точки привязки Фибо-объекта
    price1=high[70];
    time1=time[70];
    //--- координаты первой точки привязки Фибо-объекта
```

```

price2=low[50];
time2=time[50];

//--- пора создать и сам Фибо-объект
bool created=ObjectCreate(0,"Fibo",OBJ_FIBO,0,time1,price1,time2,price2);
if(created) // если объект создан удачно
{
    //--- установим цвет Фибоуровней
    ObjectSetInteger(0,"Fibo",OBJPROP_LEVELCOLOR,Blue);
    //--- кстати, а сколько у нас Фибо-уровней ?
    int levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
    Print("Fibo levels before = ",levels);
    //---выведем в Журнал=> номер уровня:значения описание_уровня
    for(int i=0;i<levels;i++)
    {
        Print(i,":",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
              " ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
    }
    //--- попробуем увеличить количество уровней на единицу
    bool modified=ObjectSetInteger(0,"Fibo",OBJPROP_LEVELS,levels+1);
    if(!modified) // не удалось количество уровней изменить
    {
        Print("Не удалось изменить количество уровней в Фибо, ошибка ",GetLastError());
    }
    //--- просто сообщим
    Print("Fibo levels after = ",ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS));
    //--- зададим значение для вновь созданного уровня
    bool added=ObjectSetDouble(0,"Fibo",OBJPROP_LEVELVALUE,levels,133);
    if(added) // удалось задать значение для уровня
    {
        Print("Удалось установить еще один уровень Fibo");
        //--- также не забыть задать описание уровня
        ObjectSetString(0,"Fibo",OBJPROP_LEVELTEXT,levels,"my level");
        ChartRedraw(0);
        //--- получим актуальное значения количества уровней в Фибо-объекте
        levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
        Print("Fibo levels after adding = ",levels);
        //--- еще раз выведем все уровни - просто чтобы убедиться
        for(int i=0;i<levels;i++)
        {
            Print(i,":",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
                  " ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
        }
    }
    else // неудача при попытке увеличить количество уровней в Фибо-объекте
    {
        Print("Не удалось установить еще один уровень Fibo. Ошибка ",GetLastError());
    }
}

```

```
}
```

**Смотри также**

[Типы объектов](#), [Свойства объектов](#)

## ObjectSetInteger

Задает значение соответствующего свойства объекта. Свойство объекта должно быть типов [datetime](#), [int](#), [color](#), [bool](#) или [char](#). Существует 2 варианта функции.

### Установка значения свойства, не имеющего модификатора

```
bool ObjectSetInteger(
    long             chart_id,      // идентификатор графика
    string           name,          // имя
    ENUM_OBJECT_PROPERTY_INTEGER prop_id, // свойство
    long             prop_value     // значение
);
```

### Установка значения свойства с указанием модификатора

```
bool ObjectSetInteger(
    long             chart_id,      // идентификатор графика
    string           name,          // имя
    ENUM_OBJECT_PROPERTY_INTEGER prop_id, // свойство
    int              prop_modifier, // модификатор
    long             prop_value     // значение
);
```

#### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя объекта.

*prop\_id*

[in] Идентификатор свойства объекта. Значение может быть одним из значений перечисления [ENUM\\_OBJECT\\_PROPERTY\\_INTEGER](#).

*prop\_modifier*

[in] Модификатор указанного свойства. Означает номер уровня в [инструментах Фибоначчи](#) и в графическом объекте Вилы Эндрюса. Нумерация уровней начинается с нуля.

*prop\_value*

[in] Значение свойства.

#### Возвращаемое значение

Возвращает `true` только в том случае, если команда на изменение свойств графического объекта успешно отправлена графику, иначе возвращает `false`. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

#### Примечание

Функция использует асинхронный вызов - это означает, что функция не дожидается выполнения команды, успешно поставленной в очередь указанного графика, а сразу же возвращает управление.

Для проверки результата выполнения на чужом графике можно использовать функцию, запрашивающую указанное свойство объекта. Но при этом следует иметь в виду, что такие функции ставятся в конец очереди команд чужого графика и дожидаются результата выполнения, то есть могут быть затратными по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

### Пример создания таблицы [Web-цветов](#)

```

//+-----+
//|                               Table of Web Colors|
//|                               Copyright 2011, MetaQuotes Software Corp |
//|                               https://www.metaquotes.net |
//+-----+
#define X_SIZE 140      // ширина объекта OBJ_EDIT
#define Y_SIZE 33       // ширина объекта OBJ_EDIT
//+-----+
//| Массив Web-цветов
//+-----+
color ExtClr[140]=
{
    clrAliceBlue,clrAntiqueWhite,clrAqua,clrAquamarine,clrAzure,clrBeige,clrBisque,clrBrown,clrBurlyWood,clrCadetBlue,clrChartreuse,clrChocolate,clrCornsilk,clrCrimson,clrCyan,clrDarkBlue,clrDarkCyan,clrDarkGoldenrod,clrDarkGray,clrDarkMagenta,clrDarkOliveGreen,clrDarkOrange,clrDarkOrchid,clrDarkRed,clrDarkSalmon,clrDarkSlateBlue,clrDarkSlateGray,clrDarkTurquoise,clrDarkViolet,clrDeepPink,clrDeepSkyBlue,clrDodgerBlue,clrFireBrick,clrFloralWhite,clrForestGreen,clrFuchsia,clrGainsboro,clrGoldenrod,clrGray,clrGreen,clrGreenYellow,clrHoneydew,clrHotPink,clrIndianRed,clrLavender,clrLavenderBlush,clrLawnGreen,clrLemonChiffon,clrLightBlue,clrLightCoral,clrLightGoldenrod,clrLightGreen,clrLightGray,clrLightPink,clrLightSalmon,clrLightSeaGreen,clrLightSlateGray,clrLightSteelBlue,clrLightYellow,clrLime,clrLimeGreen,clrLinen,clrMediumAquaMarine,clrMediumBlue,clrMediumOrchid,clrMediumPurple,clrMediumSeaGreen,clrMediumSpringGreen,clrMediumTurquoise,clrMediumVioletRed,clrMidnightBlue,clrMintCream,clrNavajoWhite,clrNavy,clrOldLace,clrOlive,clrOliveDrab,clrOrange,clrOrangeRed,clrOliveDrab,clrPaleGreen,clrPaleTurquoise,clrPaleVioletRed,clrPapayaWhip,clrPeachPuff,clrPeru,clrPlum,clrPurple,clrRed,clrRosyBrown,clrRoyalBlue,clrSaddleBrown,clrSalmon,clrSandyBrown,clrSienna,clrSilver,clrSkyBlue,clrSlateBlue,clrSlateGray,clrSnow,clrSpringGreen,clrTeal,clrThistle,clrTomato,clrTurquoise,clrViolet,clrWheat,clrWhite,clrWhiteSmoke,clrYellow
};

//+-----+
//| Создание и инициализация объекта OBJ_EDIT
//+-----+
void CreateColorBox(int x,int y,color c)
{
    //--- сгенерируем по имени цвета имя для нового объекта
    string name="ColorBox_"+(string)x+"_"+(string)y;
    //--- создадим новый объект OBJ_EDIT
}

```

```
if(!ObjectCreate(0,name,OBJ_EDIT,0,0,0))
{
    Print("Не удалось создать объект: '",name,"'");
    return;
}
//--- зададим координаты точки привязки, ширину и высоту в пикселях
ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x*X_SIZE);
ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y*Y_SIZE);
ObjectSetInteger(0,name,OBJPROP_XSIZE,X_SIZE);
ObjectSetInteger(0,name,OBJPROP_YSIZE,Y_SIZE);
//--- установим цвет текста для объекта
if(clrBlack==c) ObjectSetInteger(0,name,OBJPROP_COLOR,clrWhite);
else           ObjectSetInteger(0,name,OBJPROP_COLOR,clrBlack);
//--- установим цвет фона
ObjectSetInteger(0,name,OBJPROP_BGCOLOR,c);
//--- установим текст объекта OBJ_EDIT соответствующим цвету фона
ObjectSetString(0,name,OBJPROP_TEXT,(string)c);
}

//-----+
// | Функция запуска скрипта на выполнение |
//-----+  
void OnStart()
{
//--- создадим таблицу из цветовых блоков 7x20
for(uint i=0;i<140;i++)
    CreateColorBox(i%7,i/7,ExtClr[i]);
}
```

#### Смотри также

[Типы объектов](#), [Свойства объектов](#)

## ObjectSetString

Задает значение соответствующего свойства объекта. Свойство объекта должно быть типа [string](#). Существует 2 варианта функции.

### Установка значения свойства, не имеющего модификатора

```
bool ObjectSetString(
    long             chart_id,           // идентификатор графика
    string          name,               // имя
    ENUM_OBJECT_PROPERTY_STRING prop_id, // свойство
    string          prop_value        // значение
);
```

### Установка значения свойства с указанием модификатора

```
bool ObjectSetString(
    long             chart_id,           // идентификатор графика
    string          name,               // имя
    ENUM_OBJECT_PROPERTY_STRING prop_id, // свойство
    int              prop_modifier,     // модификатор
    string          prop_value        // значение
);
```

#### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя объекта.

*prop\_id*

[in] Идентификатор свойства объекта. Значение может быть одним из значений перечисления [ENUM\\_OBJECT\\_PROPERTY\\_STRING](#).

*prop\_modifier*

[in] Модификатор указанного свойства. Означает номер уровня в [инструментах Фибоначчи](#) и в графическом объекте Вилы Эндрюса. Нумерация уровней начинается с нуля.

*prop\_value*

[in] Значение свойства.

#### Возвращаемое значение

Возвращает true только в том случае, если команда на изменение свойств графического объекта успешно отправлена графику, иначе возвращает false. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

#### Примечание

Функция использует асинхронный вызов - это означает, что функция не дожидается выполнения команды, успешно поставленной в очередь указанного графика, а сразу же возвращает управление.

Для проверки результата выполнения на чужом графике можно использовать функцию, запрашивающую указанное свойство объекта. Но при этом следует иметь в виду, что такие функции ставятся в конец очереди команд чужого графика и дожидаются результата выполнения, то есть могут быть затратными по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

При переименовании графического объекта одновременно формируются два события, которые можно обработать в эксперте или индикаторе функцией [OnChartEvent\(\)](#):

- событие удаления объекта со старым именем;
- событие создания графического объекта с новым именем.

## ObjectGetDouble

Возвращает значение соответствующего свойства объекта. Свойство объекта должно быть типа [double](#). Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
double ObjectGetDouble(
    long             chart_id,           // идентификатор графика
    string           name,              // имя объекта
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id, // идентификатор свойства
    int              prop_modifier=0   // модификатор свойства, если требуется
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool ObjectGetDouble(
    long             chart_id,           // идентификатор графика
    string           name,              // имя объекта
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id, // идентификатор свойства
    int              prop_modifier,     // модификатор свойства
    double&          double_var        // сюда примем значение свойства
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя объекта.

*prop\_id*

[in] Идентификатор свойства объекта. Значение может быть одним из значений перечисления [ENUM\\_OBJECT\\_PROPERTY\\_DOUBLE](#).

*prop\_modifier*

[in] Модификатор указанного свойства. Для первого варианта по умолчанию значение модификатора равно 0. Большинство свойств не требуют модификатора. Означает номер уровня в [инструментах Фибоначчи](#) и в графическом объекте Вилы Эндрюса. Нумерация уровней начинается с нуля.

*double\_var*

[out] Переменная типа double, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа double для первого варианта вызова.

Для второго варианта вызова возвращает true, если данное свойство поддерживается и значение было помещено в переменную double\_var, иначе возвращает false. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

#### Примечание

Функция использует синхронный вызов - это означает, что функция дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом, и поэтому данная функция может быть затратной по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

## ObjectGetInteger

Возвращает значение соответствующего свойства объекта. Свойство объекта должно быть типов [datetime](#), [int](#), [color](#), [bool](#) или [char](#). Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
long ObjectGetInteger(
    long             chart_id,           // идентификатор графика
    string          name,               // имя объекта
    ENUM_OBJECT_PROPERTY_INTEGER prop_id, // идентификатор свойства
    int              prop_modifier=0   // модификатор свойства, если
```

2. Возвращает `true` или `false` в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool ObjectGetInteger(
    long             chart_id,           // идентификатор графика
    string          name,               // имя объекта
    ENUM_OBJECT_PROPERTY_INTEGER prop_id, // идентификатор свойства
    int              prop_modifier,     // модификатор свойства
    long&           long_var           // сюда примем значение свойства
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя объекта.

*prop\_id*

[in] Идентификатор свойства объекта. Значение может быть одним из значений перечисления [ENUM\\_OBJECT\\_PROPERTY\\_INTEGER](#).

*prop\_modifier*

[in] Модификатор указанного свойства. Для первого варианта по умолчанию значение модификатора равно 0. Большинство свойств не требуют модификатора. Означает номер уровня в [инструментах Фибоначчи](#) и в графическом объекте Вилы Эндрюса. Нумерация уровней начинается с нуля.

*long\_var*

[out] Переменная типа `long`, принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа `long` для первого варианта вызова.

Для второго варианта вызова возвращает true, если данное свойство поддерживается и значение было помещено в переменную `long_var`, иначе возвращает false. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

#### Примечание

Функция использует синхронный вызов - это означает, что функция дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом, и поэтому данная функция может быть затратной по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

## ObjectGetString

Возвращает значение соответствующего свойства объекта. Свойство объекта должно быть типа [string](#). Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
string ObjectGetString(
    long             chart_id,           // идентификатор графика
    string          name,                // имя объекта
    ENUM_OBJECT_PROPERTY_STRING prop_id, // идентификатор свойства
    int             prop_modifier=0     // модификатор свойства, если требуется
);
```

2. Возвращает true или false в зависимости от успешности выполнения функции. В случае успеха значение свойства помещается в приемную переменную, передаваемую по ссылке последним параметром.

```
bool ObjectGetString(
    long             chart_id,           // идентификатор графика
    string          name,                // имя объекта
    ENUM_OBJECT_PROPERTY_STRING prop_id, // идентификатор свойства
    int             prop_modifier,      // модификатор свойства
    string&         string_var        // сюда примем значение свойства
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*name*

[in] Имя объекта.

*prop\_id*

[in] Идентификатор свойства объекта. Значение может быть одним из значений перечисления [ENUM\\_OBJECT\\_PROPERTY\\_STRING](#).

*prop\_modifier*

[in] Модификатор указанного свойства. Для первого варианта по умолчанию значение модификатора равно 0. Большинство свойств не требуют модификатора. Означает номер уровня в [инструментах Фибоначчи](#) и в графическом объекте Вилы Эндрюса. Нумерация уровней начинается с нуля.

*string\_var*

[out] Переменная типа [string](#), принимающая значение запрашиваемого свойства.

### Возвращаемое значение

Значение типа [string](#) для первого варианта вызова.

Для второго варианта вызова возвращает true, если данное свойство поддерживается и значение было помещено в переменную string\_var, иначе возвращает false. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

#### Примечание

Функция использует синхронный вызов - это означает, что функция дожидается выполнения всех команд, которые были помещены в очередь графика перед её вызовом, и поэтому данная функция может быть затратной по времени. Нужно иметь это обстоятельство в виду, если ведется работа с большим количеством объектов на графике.

При переименовании графического объекта одновременно формируются два события, которые можно обработать в эксперте или индикаторе функцией [OnChartEvent\(\)](#):

- событие удаления объекта со старым именем;
- событие создания графического объекта с новым именем.

## TextSetFont

Устанавливает шрифт для вывода текста методами рисования и возвращает результат успешности этой операции. По умолчанию используется шрифт Arial и размер -120 (12 pt).

```
bool TextSetFont(
    const string name,           // имя шрифта или путь к файлу шрифта на диске
    int size,                   // размер шрифта
    uint flags,                 // комбинация флагов
    int orientation=0           // угол наклона текста
);
```

### Параметры

*name*

[in] Имя шрифта в системе, или имя ресурса, содержащего шрифт, или путь к файлу шрифта на диске.

*size*

[in] Размер шрифта, который может задаваться положительными и отрицательными значениями. При положительных значениях размер выводимого текста не зависит от настроек размеров шрифтов в операционной системе. При отрицательных значениях значение задается в десятых долях пункта и размер текста будет зависеть от настроек системы ("стандартный масштаб" или "крупный масштаб"). Более подробно о разнице в режимах смотрите в Примечании.

*flags*

[in] Комбинация [флагов](#), описывающих стиль шрифта.

*orientation*

[in] Угол наклона текста по горизонтали к оси X, единица измерения равна 0.1 градуса. То есть *orientation*=450 означает наклон в 45 градусов.

### Возвращаемое значение

Возвращает true в случае успешной установки текущего шрифта, иначе false. Возможные коды ошибок:

- ERR\_INVALID\_PARAMETER(4003) - *name* представляет NULL или "" (пустая строка),
- ERR\_INTERNAL\_ERROR(4001) - ошибка операционной системы (например, попытка создания несуществующего шрифта).

### Примечание

Если в имени шрифта используется ":" , то шрифт загружается из [ресурса EX5](#). Если имя шрифта *name* указано с расширением, то шрифт загружается из файла, при этом - если путь начинается с "\" или "/", то файл ищется относительно каталога MQL5, иначе ищется относительно пути EX5-файла, вызвавшего функцию `TextSetFont()`.

Размер шрифта задается положительными или отрицательными значениями, знак определяет зависимость размера текста от настроек операционной системы (масштаба шрифта).

- Если размер задается положительным числом, то при отображении логического шрифта в физический происходит преобразование размера в физические единицы измерения устройства (пиксели) и этот размер соответствует высоте ячеек символов из доступных шрифтов. Не

рекомендуется в тех случаях, когда предполагается совместное использование на графике текстов, выведенных функцией [TextOut\(\)](#), и текстов, отображаемых с помощью графического объекта [OBJ\\_LABEL](#) ("Текстовая метка").

- Если размер задается отрицательным числом, то указанный размер предполагается заданным в десятых долях логического пункта (значение -350 равно 35 логических пунктов) и делится на 10, а затем полученное значение преобразуется в физические единицы измерения устройства (пиксели) и соответствует абсолютному значению высоты символа из доступных шрифтов. Чтобы получить на экране текст такого же размера, как и в объекте [OBJ\\_LABEL](#), возьмите указанный в свойствах объекта размер шрифта и умножьте на -10.

Флаги могут использоваться в виде комбинации флагов стиля с одним из флагов, задающим толщину шрифта. Наименования флагов приведены ниже.

#### Флаги для задания стиля начертания шрифта

Флаг	Описание
FONT_ITALIC	Курсив
FONT_UNDERLINE	Подчёркивание
FONT_STRIKEOUT	Перечёркивание

#### Флаги для задания толщины шрифта

Флаг
FW_DONTCARE
FW_THIN
FW_EXTRALIGHT
FW_ULTRALIGHT
FW_LIGHT
FW_NORMAL
FW_REGULAR
FW_MEDIUM
FW_SEMIBOLD
FW_DEMIBOLD
FW_BOLD
FW_EXTRABOLD
FW_ULTRABOLD
FW_HEAVY
FW_BLACK

#### Смотри также

[Ресурсы](#), [ResourceCreate\(\)](#), [ResourceSave\(\)](#), [TextOut\(\)](#)

## TextOut

Выводит текст в пользовательский массив (буфер) и возвращает результат успешности этой операции. Данный массив предназначается для создания графического [ресурса](#).

```
bool TextOut(
    const string      text,           // выводимый текст
    int               x,              // координата X
    int               y,              // координата Y
    uint              anchor,         // способ привязки
    uint              &data[],        // буфер для вывода
    uint              width,          // ширина буфера в точках
    uint              height,         // высота буфера в точках
    uint              color,          // цвет текста
    ENUM_COLOR_FORMAT color_format  // формат цвета для вывода
);
```

### Параметры

*text*

[in] Выводимый текст, который будет записан в буфер. Осуществляется только односторонний вывод текста.

*x*

[in] Координата X точки привязки для выводимого текста.

*y*

[in] Координата Y точки привязки для выводимого текста.

*anchor*

[in] Значение из набора 9 предопределенных способов расположения точки привязки выводимого текста. Задаётся комбинацией двух флагов - флага выравнивания текста по горизонтали и флага выравнивания текста по вертикали. Наименования флагов приведены в Примечании.

*data[]*

[in] Буфер, в который выводится текст. Данный буфер используется для создания графического [ресурса](#).

*width*

[in] Ширина буфера в точках (пикселях).

*height*

[in] Высота буфера в точках (пикселях).

*color*

[in] Цвет текста.

*color\_format*

[in] Формат цвета, задаётся значением из перечисления [ENUM\\_COLOR\\_FORMAT](#).

### Возвращаемое значение

Возвращает true в случае успешного выполнения, иначе false.

### Примечание

Способ привязки, задаваемый параметром *anchor*, является комбинацией двух флагов выравнивания текста по вертикали и горизонтали. Флаги выравнивания текста по горизонтали:

- TA\_LEFT - точка привязки на левой стороне ограничивающего прямоугольника
- TA\_CENTER - точка привязки по горизонтали находится в середине ограничивающего прямоугольника
- TA\_RIGHT - точка привязки на правой стороне ограничивающего прямоугольника

Флаги выравнивания текста по вертикали:

- TA\_TOP - точка привязки на верхней стороне ограничивающего прямоугольника
- TA\_VCENTER - точка привязки по вертикали находится в середине ограничивающего прямоугольника
- TA\_BOTTOM - точка привязки на нижней стороне ограничивающего прямоугольника

Возможные комбинации флагов и задаваемые ими способы привязки показаны на рисунке.



### Пример:

```

//--- ширина и высота канваса (холста, на котором происходит рисование)
#define IMG_WIDTH 200
#define IMG_HEIGHT 200

//--- перед запуском скрипта покажем окно с параметрами
#property script_show_inputs

//--- дадим возможность задавать формат цвета
input ENUM_COLOR_FORMAT clr_format=COLOR_FORMAT_XRGB_NOALPHA;
//--- массив (буфер) для отрисовки
uint ExtImg[IMG_WIDTH*IMG_HEIGHT];

//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- создадим объект OBJ_BITMAP_LABEL для рисования
ObjectCreate(0, "CLOCK", OBJ_BITMAP_LABEL, 0, 0, 0);
//--- укажем имя графического ресурса для отрисовки в объекте CLOCK
ObjectSetString(0, "CLOCK", OBJPROP_BMPFILE, "::IMG");

//--- вспомогательные переменные

```

```

double a; // угол стрелки
uint nm=2700; // счетчик минут
uint nh=2700*12; // счетчик часов
uint w,h; // переменные для получения размеров текстовых строк
int x,y; // переменные для расчета текущих координат точки привязки тега

//--- крутим стрелки часов в бесконечном цикле, пока скрипт не остановят
while(!IsStopped())
{
    //--- очистка массива буфера рисования часов
    ArrayFill(ExtImg,0,IMG_WIDTH*IMG_HEIGHT,0);
    //--- установка шрифта для отрисовки цифр на циферблате
    TextSetFont("Arial",-200,FW_EXTRABOLD,0);
    //--- рисуем циферблат
    for(int i=1;i<=12;i++)
    {
        //--- получим размеры текущего часа на циферблате
        TextGetSize(string(i),w,h);
        //--- вычислим координаты текущего часа на циферблате
        a=-((i*300)%3600*M_PI)/1800.0;
        x=IMG_WIDTH/2-int(sin(a)*80+0.5+w/2);
        y=IMG_HEIGHT/2-int(cos(a)*80+0.5+h/2);
        //--- вывод этого часа на циферблат в буфер ExtImg[]
        TextOut(string(i),x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_black);
    }
    //--- теперь установим шрифт для отрисовки минутной стрелки
    TextSetFont("Arial",-200,FW_EXTRABOLD,-int(nm%3600));
    //--- получим размеры минутной стрелки
    TextGetSize("---->",w,h);
    //--- вычислим координаты минутной стрелки на циферблате
    a=- (nm%3600*M_PI)/1800.0;
    x=IMG_WIDTH/2-int(sin(a)*h/2+0.5);
    y=IMG_HEIGHT/2-int(cos(a)*h/2+0.5);
    //--- вывод минутной стрелки на циферблат в буфер ExtImg[]
    TextOut("---->",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_black);

    //--- теперь установим шрифт для отрисовки часовой стрелки
    TextSetFont("Arial",-200,FW_EXTRABOLD,-int(nh/12%3600));
    TextGetSize("==>",w,h);
    //--- вычислим координаты часовой стрелки на циферблате
    a=- (nh/12%3600*M_PI)/1800.0;
    x=IMG_WIDTH/2-int(sin(a)*h/2+0.5);
    y=IMG_HEIGHT/2-int(cos(a)*h/2+0.5);
    //--- вывод часовой стрелки на циферблат в буфер ExtImg[]
    TextOut("==>",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_black);

    //--- обновление графического ресурса
    ResourceCreate(":IMG",ExtImg,IMG_WIDTH,IMG_HEIGHT,0,0,IMG_WIDTH,clr_format);
    //--- принудительное обновление графика
}

```

```
ChartRedraw();

//--- увеличим счетчики часа и минут
nm+=60;
nh+=60;
//--- выдержим небольшую паузу между кадрами
Sleep(10);
}

//--- удалим объект CLOCK при завершении работы скрипта
ObjectDelete(0, "CLOCK");
//---
}
```

#### Смотри также

[Ресурсы](#), [ResourceCreate\(\)](#), [ResourceSave\(\)](#), [TextGetSize\(\)](#), [TextSetFont\(\)](#)

## TextGetSize

Возвращает ширину и высоту строки при текущих [настройках шрифта](#).

```
bool TextGetSize(
    const string      text,           // строка текста
    uint&             width,          // ширина буфера в точках
    uint&             height         // высота буфера в точках
);
```

### Параметры

*text*

[in] Стока для которой получаем длину и ширину.

*width*

[out] Входной параметр для получения ширины.

*height*

[out] Входной параметр для получения высоты.

### Возвращаемое значение

Возвращает true в случае успешного выполнения, иначе false. Возможные коды ошибок:

- ERR\_INTERNAL\_ERROR(4001) - в случае ошибки операционной системы.

### Смотри также

[Ресурсы](#), [ResourceCreate\(\)](#), [ResourceSave\(\)](#), [TextSetFont\(\)](#), [TextOut\(\)](#)

## Функции для работы с техническими индикаторами

Все функции типа `iMA`, `iAC`, `iMACD`, `iIchimoku` и т.п., создают в глобальном кеше клиентского терминала копию соответствующего технического индикатора. Если копия индикатора с этими параметрами уже существует, то новая копия не создается, а увеличивается счетчик ссылок на данную копию.

Эти функции возвращают хэндл соответствующей копии индикатора. Используя этот хэндл в дальнейшем можно получать данные, рассчитанные соответствующим индикатором. Данные соответствующего буфера (технические индикаторы содержат рассчитанные данные в своих внутренних буферах, которых, в зависимости от индикатора, может быть от 1 до 5) можно копировать в MQL5-программу при помощи функции [CopyBuffer\(\)](#).

Нельзя обратиться к данным индикатора сразу после его создания, так как на расчет значений индикатора требуется некоторое время, и поэтому создавать хэндлы индикаторов лучше всего в `OnInit()`. Функция [iCustom\(\)](#) создает соответствующий пользовательский индикатор и при успешном создании возвращает его хэндл. Пользовательские индикаторы могут содержать до 512 индикаторных буферов, содержимое которых также можно получать при помощи функции [CopyBuffer\(\)](#), используя полученный хэндл.

Существует универсальный метод создания любого технического индикатора с помощью функции [IndicatorCreate\(\)](#). Эта функция получает в качестве входных параметров:

- имя символа;
- таймфрейм;
- тип создаваемого индикатора;
- количество входных параметров индикатора;
- массив типа [MqlParam](#), содержащий все необходимые входные параметры.

Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

**Примечание.** Многократное обращение к функции индикатора с одними и теми же параметрами в пределах одной MQL5-программы не приводит к многократному увеличению счетчика ссылок, счетчик будет увеличен всего один раз на 1. Однако рекомендуется получать хэндлы индикаторов в функции [OnInit\(\)](#) или в конструкторе класса, с последующим использованием полученных хэндов в остальных функциях. Счетчик ссылок уменьшается при deinициализации MQL5-программы.

Все индикаторные функции имеют как минимум 2 параметра - символ и период. Значение символа [NULL](#) означает текущий инструмент, значение периода 0 означает текущий [таймфрейм](#).

Функция	Возвращает хэндл индикатора:
<a href="#">iAC</a>	Accelerator Oscillator
<a href="#">iAD</a>	Accumulation/Distribution
<a href="#">iADX</a>	Average Directional Index
<a href="#">iADXWilder</a>	Average Directional Index by Welles Wilder
<a href="#">iAlligator</a>	Alligator

<a href="#"><u>iAMA</u></a>	Adaptive Moving Average
<a href="#"><u>iAO</u></a>	Awesome Oscillator
<a href="#"><u>iATR</u></a>	Average True Range
<a href="#"><u>iBearsPower</u></a>	Bears Power
<a href="#"><u>iBands</u></a>	Bollinger Bands®
<a href="#"><u>iBullsPower</u></a>	Bulls Power
<a href="#"><u>iCCI</u></a>	Commodity Channel Index
<a href="#"><u>iChaikin</u></a>	Chaikin Oscillator
<a href="#"><u>iCustom</u></a>	пользовательский индикатор
<a href="#"><u>iDEMA</u></a>	Double Exponential Moving Average
<a href="#"><u>iDeMarker</u></a>	DeMarker
<a href="#"><u>iEnvelopes</u></a>	Envelopes
<a href="#"><u>iForce</u></a>	Force Index
<a href="#"><u>iFractals</u></a>	Fractals
<a href="#"><u>iFrAMA</u></a>	Fractal Adaptive Moving Average
<a href="#"><u>iGator</u></a>	Gator Oscillator
<a href="#"><u>iIchimoku</u></a>	Ichimoku Kinko Hyo
<a href="#"><u>iBWMFI</u></a>	Market Facilitation Index by Bill Williams
<a href="#"><u>iMomentum</u></a>	Momentum
<a href="#"><u>iMFI</u></a>	Money Flow Index
<a href="#"><u>iMA</u></a>	Moving Average
<a href="#"><u>iOsMA</u></a>	Moving Average of Oscillator (MACD histogram)
<a href="#"><u>iMACD</u></a>	Moving Averages Convergence-Divergence
<a href="#"><u>iOBV</u></a>	On Balance Volume
<a href="#"><u>iSAR</u></a>	Parabolic Stop And Reverse System
<a href="#"><u>iRSI</u></a>	Relative Strength Index
<a href="#"><u>iRVI</u></a>	Relative Vigor Index
<a href="#"><u>iStdDev</u></a>	Standard Deviation
<a href="#"><u>iStochastic</u></a>	Stochastic Oscillator
<a href="#"><u>iTEMA</u></a>	Triple Exponential Moving Average
<a href="#"><u>iTriX</u></a>	Triple Exponential Moving Averages Oscillator

<a href="#"><u>iWPR</u></a>	Williams' Percent Range
<a href="#"><u>iVIDyA</u></a>	Variable Index Dynamic Average
<a href="#"><u>iVolumes</u></a>	Volumes

## iAC

Создает в глобальном кеше клиентского терминала индикатор Accelerator Oscillator и возвращает его хэндл. Всего один буфер.

```
int iAC(
    string          symbol,      // имя символа
    ENUM_TIMEFRAMES period      // период
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iAC.mq5 |
//|           Copyright 2011, MetaQuotes Software Corp. |
//|           https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует, как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iAC."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- построение iAC
#property indicator_label1 "iAC"
#property indicator_type1 DRAW_COLOR_HISTOGRAM
#property indicator_color1 clrGreen, clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
```

```

//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iAC,           // использовать iAC
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iAC;           // тип функции
input string             symbol=" ";            // символ
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT; // таймфрейм
//--- индикаторные буферы
double iACBuffer[];
double iACColors[];
//--- переменная для хранения хэндла индикатора iAC
int     handle;
//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Accelerator Oscillator
int     bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
    SetIndexBuffer(0,iACBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,iACColors,INDICATOR_COLOR_INDEX);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
//--- создадим хэндл индикатора
    if(type==Call_iAC)
        handle=iAC(name,period);
    else
        handle=IndicatorCreate(name,period,IND_AC);
//--- если не удалось создать хэндл
    if(handle==INVALID_HANDLE)
}

```

```

{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iAC для пары %s/%s, код ошибки
                name,
                EnumToString(period),
                GetLastError()));

    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Accelerator Oscillator
short_name=StringFormat("iAC(%s/%s)",name,EnumToString(period));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- количество копируемых значений из индикатора iAC
    int values_to_copy;
    //--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }
    //--- если это первый запуск вычислений нашего индикатора или изменилось количество з
    //--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- если массив iACBuffer больше, чем значений в индикаторе iAC на паре symbol
        //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
}
}

```

```

//--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
//--- для расчета добавилось не более одного бара
values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массивы iACBuffer и iACColors значениями из индикатора Accelerator Oscillator
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArraysFromBuffer(iACBuffer,iACColors,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
short_name,
values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Accelerator Oscillator
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буферы из индикатора iAC |+
//+-----+
bool FillArraysFromBuffer(double &values[],           // индикаторный буфер значений Accelerator
                           double &color_indexes[], // цветовой буфер (для хранения индексов цветов)
                           int ind_handle,          // хэндл индикатора iAC
                           int amount                // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iACBuffer значениями из индикаторного буфера под индексом 0
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iAC, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться не готовым
return(false);
}

//--- теперь копируем индексы цветов
if(CopyBuffer(ind_handle,1,0,amount,color_indexes)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать значения цветов из индикатора iAC, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться не готовым
return(false);
}

//--- все получилось
return(true);
}

```

```
//+-----+
//| Indicator deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iAD

Возвращает хэндл индикатора Accumulation/Distribution. Всего один буфер.

```
int iAD(
    string           symbol,          // имя символа
    ENUM_TIMEFRAMES period,         // период
    ENUM_APPLIED_VOLUME applied_volume // тип объема для расчета
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*applied\_volume*

[in] Используемый объем. Может быть любой из [ENUM\\_APPLIED\\_VOLUME](#).

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iAD.mq5 |
//|           Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iAD."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iAD
#property indicator_label1  "iAD"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Переисчисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iAD,           // использовать iAD
    Call_IndicatorCreate // использовать IndicatorCreate
};
//--- входные параметры
input Creation          type=Call_iAD;           // тип функции
input ENUM_APPLIED_VOLUME volumes;                // используемый объем
input string              symbol=" ";             // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT; // таймфрейм
//--- индикаторный буфер
double                  iADBuffer[];
//--- переменная для хранения хэндла индикатора iAD
int                     handle;
//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Accumulation/Distribution
int      bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iADBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=_Symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
//--- создадим хэндл индикатора
    if(type==Call_iAD)
        handle=iAD(name,period,volumes);
    else
    {
//--- заполним структуру значениями параметров индикатора

```

```

MqlParam pars[1];
pars[0].type=TYPE_INT;
pars[0].integer_value=volumes;
handle=IndicatorCreate(name,period,IND_AD,1,pars);
}

//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
//--- сообщим о неудаче и выведем номер ошибки
PrintFormat("Не удалось создать хэндл индикатора iAD для пары %s/%s, код ошибки
name,
EnumToString(period),
GetLastError()));

//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Accumulation/Distribution
short_name=StringFormat("iAD(%s/%s)",name,EnumToString(period));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iAD
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
}
}

```

```

//--- если массив iADBuffer больше, чем значений в индикаторе iAD на паре symbol
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
if(calculated>rates_total) values_to_copy=rates_total;
else values_to_copy=calculated;
}
else
{
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
//--- для расчета добавилось не более одного бара
values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массив iADBuffer значениями из индикатора Accumulation/Distribution
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем расчет
if(!FillArrayFromBuffer(iADBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
short_name,
values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Accumulation/Distribution
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буфера из индикатора iAD |+
//+-----+
bool FillArrayFromBuffer(double &values[], // индикаторный буфер линии Accumulation,
int ind_handle, // хэндл индикатора iAD
int amount // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();
//--- заполняем часть массива iADBuffer значениями из индикаторного буфера под индексом 0
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iAD, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считать, что
return(false);
}
//--- все получилось
return(true);
}
//+-----+
//| Indicator deinitialization function |+

```

```
//+-----+
void OnDeinit(const int reason)
{
//--- ПОЧИСТИМ график при удалении индикатора
Comment("");
}
```

## iADX

Возвращает хэндл индикатора Average Directional Movement Index.

```
int iADX(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,         // период
    int             adx_period       // период усреднения
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*adx\_period*

[in] Период для вычисления индекса.

### Примечание

Номера буферов: 0 - MAIN\_LINE, 1 - PLUSDI\_LINE, 2 - MINUSDI\_LINE.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iADX.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iADX."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_plots   3
```

```

//--- plot ADX
#property indicator_label1 "ADX"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrLightSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1

//--- plot DI_plus
#property indicator_label2 "DI_plus"
#property indicator_type2 DRAW_LINE
#property indicator_color2 clrYellowGreen
#property indicator_style2 STYLE_SOLID
#property indicator_width2 1

//--- plot DI_minus
#property indicator_label3 "DI_minus"
#property indicator_type3 DRAW_LINE
#property indicator_color3 clrWheat
#property indicator_style3 STYLE_SOLID
#property indicator_width3 1

//+-----+
// | Перечисление способов создания хэндла |
//+-----+

enum Creation
{
    Call_iADX, // использовать iADX
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation type=Call_iADX; // тип функции
input int adx_period=14; // период расчета
input string symbol=" "; // символ
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // таймфрейм

//--- индикаторные буферы
double ADXBuffer[];
double DI_plusBuffer[];
double DI_minusBuffer[];

//--- переменная для хранения хэндла индикатора iADX
int handle;

//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Average Directional Movement Index
int bars_calculated=0;

//+-----+
// | Custom indicator initialization function |
//+-----+

int OnInit()
{
    //--- привязка массивов к индикаторным буферам
}

```

```

SetIndexBuffer(0,ADXBuffer,INDICATOR_DATA);
SetIndexBuffer(1,DI_plusBuffer,INDICATOR_DATA);
SetIndexBuffer(2,DI_minusBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
name=symbol;
//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iADX)
    handle=iADX(name,period,adx_period);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[1];
    pars[0].type=TYPE_INT;
    pars[0].integer_value=adx_period;
    handle=IndicatorCreate(name,period,IND_ADX,1,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iADX для пары %s/%s, код ошибки",
               name,
               EnumToString(period),
               GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Average Directional Movement
short_name=StringFormat("iADX(%s/%s period=%d)",name,EnumToString(period),adx_period);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

+-----+
//| Custom indicator iteration function |
+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
```

```

        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {

//--- количество копируемых значений из индикатора iADX
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }

//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений в индикаторе
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- если массив ADXBuffer больше, чем значений в индикаторе iADX на паре symbol
        //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
        //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
        //--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }

//--- заполняем массив значениями из индикатора Average Directional Movement Index
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем работу
    if(!FillArraysFromBuffers(ADXBuffer,DI_plusBuffer,DI_minusBuffer,handle,values_to_copy))
//--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);

//--- выведем на график служебное сообщение
    Comment(comm);
//--- запомним количество значений в индикаторе Average Directional Movement Index
    bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
    return(rates_total);
}

//-----+
//| Заполняем индикаторные буфера из индикатора iADX |
//-----+

```

```

bool FillArraysFromBuffers(double &adx_values[],           // индикаторный буфер линии ADX
                           double &DIplus_values[],      // индикаторный буфер для DI+
                           double &DIminus_values[],    // индикаторный буфер для DI-
                           int ind_handle,              // хэндл индикатора iADXWilder
                           int amount                   // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива ADXBuffer значениями из индикаторного буфера под индексом 0
if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iADX, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться удаленным
return(false);
}

//--- заполняем часть массива DI_plusBuffer значениями из индикаторного буфера под индексом 1
if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iADX, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться удаленным
return(false);
}

//--- заполняем часть массива DI_minusBuffer значениями из индикаторного буфера под индексом 2
if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iADX, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться удаленным
return(false);
}

//--- все получилось
return(true);
}

//-----+
//| Indicator deinitialization function
//-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}

```

## iADXWilder

Возвращает хэндл индикатора Average Directional Movement Index by Welles Wilder.

```
int iADXWilder(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,         // период
    int             adx_period       // период усреднения
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*adx\_period*

[in] Период для вычисления индекса.

### Примечание

Номера буферов: 0 - MAIN\_LINE, 1 - PLUSDI\_LINE, 2 - MINUSDI\_LINE.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                                         iADXWilder.mq5 |
//|                                         Copyright 2011, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iADXWilder."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_plots   3
```

```

//--- plot ADX
#property indicator_label1 "ADX"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrLightSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1

//--- plot DI_plus
#property indicator_label2 "DI_plus"
#property indicator_type2 DRAW_LINE
#property indicator_color2 clrYellowGreen
#property indicator_style2 STYLE_SOLID
#property indicator_width2 1

//--- plot DI_minus
#property indicator_label3 "DI_minus"
#property indicator_type3 DRAW_LINE
#property indicator_color3 clrWheat
#property indicator_style3 STYLE_SOLID
#property indicator_width3 1

//+-----+
// | Перечисление способов создания хэндла |
//+-----+

enum Creation
{
    Call_iADXWildler,           // использовать iADXWildler
    Call_IndicatorCreate         // использовать IndicatorCreate
};

//--- входные параметры
input Creation             type=Call_iADXWildler;      // тип функции
input int                   adx_period=14;            // период расчета
input string                symbol=" ";               // символ
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;   // таймфрейм

//--- индикаторные буферы
double          ADXBuffer[];
double          DI_plusBuffer[];
double          DI_minusBuffer[];

//--- переменная для хранения хэндла индикатора iADXWildler
int      handle;

//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Average Directional Movement Index
int      bars_calculated=0;

//+-----+
// | Custom indicator initialization function |
//+-----+

int OnInit()
{
    //--- привязка массивов к индикаторным буферам

```

```

SetIndexBuffer(0,ADXBuffer,INDICATOR_DATA);
SetIndexBuffer(1,DI_plusBuffer,INDICATOR_DATA);
SetIndexBuffer(2,DI_minusBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
name=symbol;
//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iADXWilder)
    handle=iADXWilder(name,period,adx_period);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[1];
    pars[0].type=TYPE_INT;
    pars[0].integer_value=adx_period;
    handle=IndicatorCreate(name,period,IND_ADXW,1,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iADXWilder для пары %s/%s, код
                name,
                EnumToString(period),
                GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Average Directional N
short_name=StringFormat("iADXWilder(%s/%s period=%d)",name,EnumToString(period),adx_period);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

+-----+
//| Custom indicator iteration function
+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
```

```

        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
//--- количество копируемых значений из индикатора iADXWilder
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError
        return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
    {
        //--- если массив ADXBuffer больше, чем значений в индикаторе iADXWilder на пар
        //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
        //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
        //--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- заполняем массив значениями из индикатора Average Directional Movement Index by
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем раб
    if(!FillArraysFromBuffers(ADXBuffer,DI_plusBuffer,DI_minusBuffer,handle,values_to_c
//--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
        TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
        short_name,
        values_to_copy);
//--- выведем на график служебное сообщение
    Comment(comm);
//--- запомним количество значений в индикаторе Average Directional Movement Index
    bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
    return(rates_total);
}
//-----+
//| Заполняем индикаторные буфера из индикатора iADXWilder |
//-----+

```

```

bool FillArraysFromBuffers(double &adx_values[],           // индикаторный буфер линии ADX
                           double &DIplus_values[],      // индикаторный буфер для DI+
                           double &DIminus_values[],    // индикаторный буфер для DI-
                           int ind_handle,              // хэндл индикатора iADXWilder
                           int amount                   // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива ADXBuffer значениями из индикаторного буфера под индексом 0
if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iADXWilder, код ошибки %d", GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться отсутствующим
return(false);
}

//--- заполняем часть массива DI_plusBuffer значениями из индикаторного буфера под индексом 1
if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iADXWilder, код ошибки %d", GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться отсутствующим
return(false);
}

//--- заполняем часть массива DI_minusBuffer значениями из индикаторного буфера под индексом 2
if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iADXWilder, код ошибки %d", GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться отсутствующим
return(false);
}

//--- все получилось
return(true);
}

//-----+
//| Indicator deinitialization function
//-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}

```

## iAlligator

Возвращает хэндл индикатора Alligator.

```
int iAlligator(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             jaw_period,       // период для расчета челюстей
    int             jaw_shift,         // смещение челюстей по горизонтали
    int             teeth_period,      // период для расчета зубов
    int             teeth_shift,        // смещение зубов по горизонтали
    int             lips_period,       // период для расчета губ
    int             lips_shift,         // смещение губ по горизонтали
    ENUM_MA_METHOD ma_method,        // тип склаживания
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*jaw\_period*

[in] Период усреднения синей линии (челюсти аллигатора).

*jaw\_shift*

[in] Смещение синей линии относительно графика цены.

*teeth\_period*

[in] Период усреднения красной линии (зубов аллигатора).

*teeth\_shift*

[in] Смещение красной линии относительно графика цены.

*lips\_period*

[in] Период усреднения зеленой линии (губ аллигатора).

*lips\_shift*

[in] Смещение зеленой линии относительно графика цены.

*ma\_method*

[in] Метод усреднения. Может быть любым из значений перечисления [ENUM\\_MA\\_METHOD](#).

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хэндлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает **INVALID\_HANDLE**. Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Примечание

Номера буферов: 0 - GATORJAW\_LINE, 1 - GATORTEETH\_LINE, 2 - GATORLIPS\_LINE.

### Пример:

```
//+-----+
//|                               Demo_iAlligator.mq5 |
//|           Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iAlligator."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "здаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном Аллигаторе."

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots    3
//--- plot Jaws
#property indicator_label1  "Jaws"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- plot Teeth
#property indicator_label2  "Teeth"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- plot Lips
#property indicator_label3  "Lips"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrLime
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
```

```

//+-----+
enum Creation
{
    Call_iAlligator,           // использовать iAlligator
    Call_IndicatorCreate       // использовать IndicatorCreate
};

//--- входные параметры
input Creation             type=Call_iAlligator;   // тип функции
input string                symbol=" ";                 // символ
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT; // таймфрейм
input int                   jaw_period=13;            // период для линии Челюстей
input int                   jaw_shift=8;              // смещение линии Челюстей
input int                   teeth_period=8;           // период для линии Зубов
input int                   teeth_shift=5;            // смещение линии Челюстей
input int                   lips_period=5;            // период для линии Губ
input int                   lips_shift=3;             // смещение линии Губ
input ENUM_MA_METHOD        MA_method=MODE_SMMA;     // метод усреднения линий Аллигатор
input ENUM_APPLIED_PRICE    applied_price=PRICE_MEDIAN; // тип цены, от которой строится

//--- индикаторные буферы
double         JawsBuffer[];
double         TeethBuffer[];
double         LipsBuffer[];

//--- переменная для хранения хэндла индикатора iAlligator
int          handle;

//--- переменная для хранения
string        name=symbol;

//--- имя индикатора на графике
string        short_name;

//--- будем хранить количество значений в индикаторе Alligator
int          bars_calculated=0;
//+-----+
//| Custom indicator initialization function | 
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
    SetIndexBuffer(0,JawsBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,TeethBuffer,INDICATOR_DATA);
    SetIndexBuffer(2,LipsBuffer,INDICATOR_DATA);

//--- зададим смещение для каждой линии
    PlotIndexSetInteger(0,PLOT_SHIFT,jaw_shift);
    PlotIndexSetInteger(1,PLOT_SHIFT,teeth_shift);
    PlotIndexSetInteger(2,PLOT_SHIFT,lips_shift);

//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
}

```

```

if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}

//--- создадим хэндл индикатора
if(type==Call_iAlligator)
    handle=iAlligator(name,period,jaw_period,jaw_shift,teeth_period,
                       teeth_shift,lips_period,lips_shift,MA_method,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[8];
    //--- периоды и смещения линий Аллигатора
    pars[0].type=TYPE_INT;
    pars[0].integer_value=jaw_period;
    pars[1].type=TYPE_INT;
    pars[1].integer_value=jaw_shift;
    pars[2].type=TYPE_INT;
    pars[2].integer_value=teeth_period;
    pars[3].type=TYPE_INT;
    pars[3].integer_value=teeth_shift;
    pars[4].type=TYPE_INT;
    pars[4].integer_value=lips_period;
    pars[5].type=TYPE_INT;
    pars[5].integer_value=lips_shift;
    //--- тип сглаживания
    pars[6].type=TYPE_INT;
    pars[6].integer_value=MA_method;
    //--- тип цены
    pars[7].type=TYPE_INT;
    pars[7].integer_value=applied_price;
//--- создадим хэндл
    handle=IndicatorCreate(name,period,IND_ALLIGATOR,8,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iAlligator для пары %s/%s, код
               name,
               EnumToString(period),
               GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Alligator
short_name=StringFormat("iAlligator(%s/%s, %d,%d,%d,%d,%d,%d)",name,EnumToString(pe
                           jaw_period,jaw_shift,teeth_period,teeth_shift,lips_period,);

```

```

IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function | 
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iAlligator
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
    return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з...
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив JawsBuffer больше, чем значений в индикаторе iAlligator на пар...
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего...
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}
//--- заполняем массивы значениями из индикатора Alligator
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем раб...
if(!FillArraysFromBuffers(JawsBuffer,jaw_shift,TeethBuffer,teeth_shift,LipsBuffer,I...
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                         TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                         short_name,

```

```

        values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Alligator
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буферы из индикатора iAlligator |+
//+-----+
bool FillArraysFromBuffers(double &jaws_buffer[], // индикаторный буфер для линии Челюстей
                           int j_shift,           // смещение линии Челюстей
                           double &teeth_buffer[], // индикаторный буфер для линии Зубов
                           int t_shift,           // смещение линии Зубов
                           double &lips_buffer[], // индикаторный буфер для линии Губ
                           int l_shift,           // смещение линии Губ
                           int ind_handle,        // хэндл индикатора iAlligator
                           int amount              // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();
//--- заполняем часть массива JawsBuffer значениями из индикаторного буфера под индексом 0
if(CopyBuffer(ind_handle,0,-j_shift,amount,jaws_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iAlligator, код ошибки %d", GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться пустым
return(false);
}

//--- заполняем часть массива TeethBuffer значениями из индикаторного буфера под индексом 1
if(CopyBuffer(ind_handle,1,-t_shift,amount,teeth_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iAlligator, код ошибки %d", GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться пустым
return(false);
}

//--- заполняем часть массива LipsBuffer значениями из индикаторного буфера под индексом 2
if(CopyBuffer(ind_handle,2,-l_shift,amount,lips_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iAlligator, код ошибки %d", GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться пустым
return(false);
}
}

```

```
//--- все получилось
    return(true);
}
//+-----+
//| Indicator deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iAMA

Возвращает хэндл индикатора Adaptive Moving Average. Всего один буфер.

```
int iAMA(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ama_period,       // период AMA
    int             fast_ma_period,   // период быстрой скользящей
    int             slow_ma_period,   // период медленной скользящей
    int             ama_shift,        // смещение индикатора по горизонтали
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ama\_period*

[in] Период вычисления, на котором вычисляется коэффициент эффективности.

*fast\_ma\_period*

[in] Быстрый период для вычисления коэффициента сглаживания в моменты быстрого рынка.

*slow\_ma\_period*

[in] Медленный период для вычисления коэффициента сглаживания в отсутствии тренда.

*ama\_shift*

[in] Сдвиг индикатора относительно ценового графика.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iAMA.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
```

```

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iAMA."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартной AMA."

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots    1
//--- plot iAMA
#property indicator_label1  "iAMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iAMA,           // использовать iAMA
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iAMA;           // тип функции
input string              symbol=" ";             // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT; // таймфрейм
input int                 ama_period=15;          // период для расчета
input int                 fast_ma_period=2;        // период быстрой скользящей
input int                 slow_ma_period=30;       // период медленной скользящей
input int                 ama_shift=0;            // смещение по горизонтали
input ENUM_APPLIED_PRICE applied_price;         // тип цены
//--- индикаторный буфер
double                  iAMABuffer[];

//--- переменная для хранения хэндла индикатора iAMA
int        handle;
//--- переменная для хранения
string     name=symbol;
//--- имя индикатора на графике
string     short_name;
//--- будем хранить количество значений в индикаторе Adaptive Moving Average
int        bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+

```

```

int OnInit()
{
//--- привязка массива к индикаторному буферу
SetIndexBuffer(0,iAMABuffer,INDICATOR_DATA);
//--- зададим смещение
PlotIndexSetInteger(0,PLOT_SHIFT,ama_shift);
//--- определимся с символом, на котором строится индикатор
name=symbol;
//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
//--- возьмем символ с графика, на котором запущен индикатор
name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iAMA)
handle=iAMA(name,period,ama_period,fast_ma_period,slow_ma_period,ama_shift,applied_price);
else
{
//--- заполним структуру значениями параметров индикатора
MqlParam pars[5];
pars[0].type=TYPE_INT;
pars[0].integer_value=ama_period;
pars[1].type=TYPE_INT;
pars[1].integer_value=fast_ma_period;
pars[2].type=TYPE_INT;
pars[2].integer_value=slow_ma_period;
pars[3].type=TYPE_INT;
pars[3].integer_value=ama_shift;
//--- тип цены
pars[4].type=TYPE_INT;
pars[4].integer_value=applied_price;
handle=IndicatorCreate(name,period,IND_AMA,5,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
//--- сообщим о неудаче и выведем номер ошибки
PrintFormat("Не удалось создать хэндл индикатора iAMA для пары %s/%s, код ошибки %d",name,
EnumToString(period),
GetLastError());
}
//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Adaptive Moving Average

```

```

short_name=StringFormat("iAMA(%s/%s,%d,%d,%d,d)",name,EnumToString(period),ama_peri
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iAMA
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError
        return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
//--- если массив iAMABuffer больше, чем значений в индикаторе iAMA на паре symb
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
//--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- заполняем массивы значениями из индикатора Adaptive Moving Average
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем раб
    if(!FillArrayFromBuffer(iAMABuffer,ama_shift,handle,values_to_copy)) return(0);
//--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),

```

```

        short_name,
        values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Adaptive Moving Average
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iAMA           |
//+-----+
bool FillArrayFromBuffer(double &ama_buffer[], // индикаторный буфер линии AMA
                         int a_shift,          // смещение линии AMA
                         int ind_handle,       // хэндл индикатора iAMA
                         int amount            // количество копируемых значений
                        )
{
//--- сбросим код ошибки
ResetLastError();
//--- заполняем часть массива iAMABuffer значениями из индикаторного буфера под индексом
if(CopyBuffer(ind_handle,0,-a_shift,amount,ama_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iAMA, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться удаленным
return(false);
}
//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function                          |
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}

```

## iAO

Возвращает хэндл индикатора Awesome Oscillator. Всего один буфер.

```
int iAO(
    string          symbol,      // имя символа
    ENUM_TIMEFRAMES period      // период
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iAO.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iAO."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- построение iAO
#property indicator_label1  "iAO"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen,clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+-----+
```

```

//| Перечисление способов создания хэндла
//+-----+
enum Creation
{
    Call_iAO,           // использовать iAO
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iAO;           // тип функции
input string             symbol=" ";           // символ
input ENUM_TIMEFRAMES   period=PERIOD_CURRENT; // таймфрейм

//--- индикаторные буферы
double      iAOBuffer[];
double      iAOColors[];

//--- переменная для хранения хэндла индикатора iAO
int        handle;

//--- переменная для хранения
string     name=symbol;
//--- имя индикатора на графике
string     short_name;
//--- будем хранить количество значений в индикаторе Awesome Oscillator
int        bars_calculated=0;
//+-----+
//| Custom indicator initialization function
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
    SetIndexBuffer(0,iAOBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,iAOColors,INDICATOR_COLOR_INDEX);
//--- определимся с символом, на котором строится индикатор
    name=_Symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
//--- создадим хэндл индикатора
    if(type==Call_iAO)
        handle=iAO(name,period);
    else
        handle=IndicatorCreate(name,period,IND_AO);
//--- если не удалось создать хэндл
    if(handle==INVALID_HANDLE)
    {
}

```

```

//--- сообщим о неудаче и выведем номер ошибки
PrintFormat("Не удалось создать хэндл индикатора iAO для пары %s/%s, код ошибки
           name,
           EnumToString(period),
           GetLastError()));

//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Awesome Oscillator
short_name=StringFormat("iAO(%s/%s)",name,EnumToString(period));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iAO
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
//--- если массив iAOBuffer больше, чем значений в индикаторе iAO на паре symbol
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else                         values_to_copy=calculated;
    }
    else
    {
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнег
}
}

```

```

//--- для расчета добавилось не более одного бара
values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массивы iAOBuffer и iAOColors значениями из индикатора Awesome Oscillator
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArraysFromBuffer(iAOBuffer,iAOColors,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
short_name,
values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Awesome Oscillator
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буферы из индикатора iAO | 
//+-----+
bool FillArraysFromBuffer(double &values[],           // индикаторный буфер значений Awesome
                           double &color_indexes[], // цветовой буфер (для хранения индексов цветов)
                           int ind_handle,          // хэндл индикатора iAO
                           int amount                // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iAOBuffer значениями из индикаторного буфера под индексом 0
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iAO, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться не готовым
return(false);
}

//--- теперь копируем индексы цветов
if(CopyBuffer(ind_handle,1,0,amount,color_indexes)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать значения цветов из индикатора iAO, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться не готовым
return(false);
}

//--- все получилось
return(true);
}

//+-----+

```

```
//| Indicator deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iATR

Возвращает хэндл индикатора Average True Range. Всего один буфер.

```
int iATR(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,         // период
    int             ma_period       // период усреднения
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения для вычисления индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iATR.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iATR."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iATR
#property indicator_label1  "iATR"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Переисчисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iATR, // использовать iATR
    Call_IndicatorCreate // использовать IndicatorCreate
};
//--- входные параметры
input int atr_period=14; // период для вычисления
input Creation type=Call_iATR; // тип функции
input string symbol=" "; // символ
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // таймфрейм
//--- индикаторный буфер
double iATRBuffer[];
//--- переменная для хранения хэндла индикатора iAC
int handle;
//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Average True Range
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iATRBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
//--- создадим хэндл индикатора
    if(type==Call_iATR)
        handle=iATR(name,period,atr_period);
    else
    {
//--- заполним структуру значениями параметров индикатора

```

```

MqlParam pars[1];
pars[0].type=TYPE_INT;
pars[0].integer_value=atr_period;
handle=IndicatorCreate(name,period,IND_ATR,1,pars);
}

//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
//--- сообщим о неудаче и выведем номер ошибки
PrintFormat("Не удалось создать хэндл индикатора iATR для пары %s/%s, код ошибки
            name,
            EnumToString(period),
            GetLastError()));

//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Average True Range
short_name=StringFormat("iATR(%s/%s, period=%d)",name,EnumToString(period),atr_peri
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iATR
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
}

```

```

//--- если массив iATRBuffer больше, чем значений в индикаторе iATR на паре символов
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
if(calculated>rates_total) values_to_copy=rates_total;
else values_to_copy=calculated;
}
else
{
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
//--- для расчета добавилось не более одного бара
values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массив iATRBuffer значениями из индикатора Average True Range
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(iATRBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
short_name,
values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Average True Range
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iATR |+
//+-----+
bool FillArrayFromBuffer(double &values[], // индикаторный буфер значений ATR
int ind_handle, // хэндл индикатора iATR
int amount // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iATRBuffer значениями из индикаторного буфера под индексом 0
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iATR, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться не готовым
return(false);
}

//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function |+

```

```
//+-----+
void OnDeinit(const int reason)
{
//--- ПОЧИСТИМ график при удалении индикатора
Comment("");
}
```

## iBearsPower

Возвращает хэндл индикатора Bears Power. Всего один буфер.

```
int iBearsPower(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения для вычисления индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iBearsPower.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iBearsPower."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- построение iBearsPower
#property indicator_label1  "iBearsPower"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
```

```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Переисчисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iBearsPower,           // использовать iBearsPower
    Call_IndicatorCreate        // использовать IndicatorCreate
};
//--- входные параметры
input Creation              type=Call_iBearsPower;   // тип функции
input int                     ma_period=13;          // период скользящей
input string                  symbol=" ";            // символ
input ENUM_TIMEFRAMES        period=PERIOD_CURRENT; // таймфрейм
//--- индикаторный буфер
double                      iBearsPowerBuffer[];
//--- переменная для хранения хэндла индикатора iBearsPower
int                          handle;
//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Bears Power
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iBearsPowerBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
//--- создадим хэндл индикатора
    if(type==Call_iBearsPower)
        handle=iBearsPower(name,period,ma_period);
    else
    {
//--- заполним структуру значениями параметров индикатора
}
}

```

```

MqlParam pars[1];
//--- период скользящей
pars[0].type=TYPE_INT;
pars[0].integer_value=ma_period;
handle=IndicatorCreate(name,period,IND_BEARS,1,pars);
}

//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
PrintFormat("Не удалось создать хэндл индикатора iBearsPower для пары %s/%s, код ошибки %d",name,
EnumToString(period),
GetLastError());
}

//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Bears Power
short_name=StringFormat("iBearsPower(%s/%s, period=%d)",name,EnumToString(period),r
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iBearsPower
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
    return(0);
}

//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)

```

```

{
    //--- если массив iBearsPowerBuffer больше, чем значений в индикаторе iBearsPower
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массив iBearsPowerBuffer значениями из индикатора Bears Power
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(iBearsPowerBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Bears Power
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iBearsPower |+
//+-----+
bool FillArrayFromBuffer(double &values[], // индикаторный буфер значений Bears Power
                        int ind_handle, // хэндл индикатора iBearsPower
                        int amount // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iBearsPowerBuffer значениями из индикаторного буфера подряд
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iBearsPower, код ошибки %d", GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считать
    return(false);
}
//--- все получилось
return(true);
}

//+-----+

```

```
//| Indicator deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iBands

Возвращает хэндл индикатора Bollinger Bands®.

```
int iBands(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             bands_period,     // период для расчета средней линии
    int             bands_shift,      // смещение индикатора по горизонтали
    double          deviation,        // кол-во стандартных отклонений
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*bands\_period*

[in] Период усреднения основной линии индикатора.

*bands\_shift*

[in] Сдвиг индикатора относительно ценового графика.

*deviation*

[in] Отклонение от основной линии.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Примечание

Номера буферов: 0 - BASE\_LINE, 1 - UPPER\_BAND, 2 - LOWER\_BAND

### Пример:

```
//+-----+
//|                               Demo_iBands.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
```

## Технические индикаторы

```

#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iBands."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots 3
//--- построение Upper
#property indicator_label1 "Upper"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrMediumSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- построение Lower
#property indicator_label2 "Lower"
#property indicator_type2 DRAW_LINE
#property indicator_color2 clrMediumSeaGreen
#property indicator_style2 STYLE_SOLID
#property indicator_width2 1
//--- построение Middle
#property indicator_label3 "Middle"
#property indicator_type3 DRAW_LINE
#property indicator_color3 clrMediumSeaGreen
#property indicator_style3 STYLE_SOLID
#property indicator_width3 1
//+-----+
// | Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iBands,           // использовать iBands
    Call_IndicatorCreate   // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iBands;           // тип функции
input int                bands_period=20;           // период скользящей средней
input int                bands_shift=0;            // сдвиг
input double              deviation=2.0;           // кол-во стандартных отклонений
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string              symbol=" ";             // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT; // таймфрейм

//--- индикаторные буферы
double                  UpperBuffer[];
double                  LowerBuffer[];

```

```

double MiddleBuffer[];
//--- переменная для хранения хэндла индикатора iBands
int handle;
//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Bollinger Bands
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
SetIndexBuffer(0,UpperBuffer,INDICATOR_DATA);
SetIndexBuffer(1,LowerBuffer,INDICATOR_DATA);
SetIndexBuffer(2,MiddleBuffer,INDICATOR_DATA);
//--- зададим смещение для каждой линии
PlotIndexSetInteger(0,PLOT_SHIFT,bands_shift);
PlotIndexSetInteger(1,PLOT_SHIFT,bands_shift);
PlotIndexSetInteger(2,PLOT_SHIFT,bands_shift);
//--- определимся с символом, на котором строится индикатор
name=symbol;
//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
//--- возьмем символ с графика, на котором запущен индикатор
name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iBands)
    handle=iBands(name,period,bands_period,bands_shift,deviation,applied_price);
else
{
//--- заполним структуру значениями параметров индикатора
MqlParam pars[4];
//--- период скользящей
pars[0].type=TYPE_INT;
pars[0].integer_value=bands_period;
//--- смещение
pars[1].type=TYPE_INT;
pars[1].integer_value=bands_shift;
//--- количество стандартных отклонений
pars[2].type=TYPE_DOUBLE;
pars[2].double_value=deviation;
}

```

```

//--- тип цены
pars[3].type=TYPE_INT;
pars[3].integer_value=applied_price;
handle=IndicatorCreate(name,period,IND_BANDS,4,pars);
}

//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
PrintFormat("Не удалось создать хэндл индикатора iBands для пары %s/%s, код ошибки %d",name,
EnumToString(period),
GetLastError());
}

//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Bollinger Bands
short_name=StringFormat("iBands(%s/%s, %d,%d,%G,%s)",name,EnumToString(period),
bands_period,bands_shift,deviation,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- количество копируемых значений из индикатора iBands
    int values_to_copy;
    //--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
        return(0);
    }

    //--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
    //--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)

```

```

{
    //--- если размер индикаторных массивов больше, чем значений в индикаторе iBands
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массив значениями из индикатора Bollinger Bands
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArraysFromBuffers(MiddleBuffer,UpperBuffer,LowerBuffer,bands_shift,handle,values_to_copy))
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Bollinger Bands
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буфера из индикатора iBands |+
//+-----+
bool FillArraysFromBuffers(double &base_values[],           // индикаторный буфер средней линии
                           double &upper_values[],          // индикаторный буфер верхней границы
                           double &lower_values[],          // индикаторный буфер нижней границы
                           int shift,                      // смещение
                           int ind_handle,                 // хэндл индикатора iBands
                           int amount                       // количество копируемых значений
                           )
{
    //--- сбросим код ошибки
    ResetLastError();
    //--- заполняем часть массива MiddleBuffer значениями из индикаторного буфера под индикатором
    if(CopyBuffer(ind_handle,0,-shift,amount,base_values)<0)
    {
        //--- если копирование не удалось, сообщим код ошибки
        PrintFormat("Не удалось скопировать данные из индикатора iBands, код ошибки %d",
                   GetLastError());
        //--- завершим с нулевым результатом - это означает, что индикатор будет считать
        return(false);
    }
}

```

```
//--- заполняем часть массива UpperBuffer значениями из индикаторного буфера под индексом shift
if(CopyBuffer(ind_handle,1,-shift,amount,upper_values)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iBands, код ошибки %d",
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться неактивным
    return(false);
}

//--- заполняем часть массива LowerBuffer значениями из индикаторного буфера под индексом shift
if(CopyBuffer(ind_handle,2,-shift,amount,lower_values)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iBands, код ошибки %d",
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться неактивным
    return(false);
}

//--- все получилось
return(true);
}

//-----+
//| Indicator deinitialization function           |
//-----+  
+-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iBullsPower

Возвращает хэндл индикатора Bulls Power. Всего один буфер.

```
int iBullsPower(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения для вычисления индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iBullsPower.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iBullsPower."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- построение iBullsPower
#property indicator_label1  "iBullsPower"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
```

```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Переисчисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iBullsPower,           // использовать iBullsPower
    Call_IndicatorCreate        // использовать IndicatorCreate
};
//--- входные параметры
input Creation              type=Call_iBullsPower;   // тип функции
input int                     ma_period=13;          // период скользящей
input string                  symbol=" ";           // символ
input ENUM_TIMEFRAMES        period=PERIOD_CURRENT; // таймфрейм
//--- индикаторный буфер
double                      iBullsPowerBuffer[];
//--- переменная для хранения хэндла индикатора iBullsPower
int                          handle;
//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Bulls Power
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iBullsPowerBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
//--- создадим хэндл индикатора
    if(type==Call_iBullsPower)
        handle=iBullsPower(name,period,ma_period);
    else
    {
//--- заполним структуру значениями параметров индикатора

```

```

MqlParam pars[1];
//--- период скользящей
pars[0].type=TYPE_INT;
pars[0].integer_value=ma_period;
handle=IndicatorCreate(name,period,IND_BULLS,1,pars);
}

//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iBullsPower для пары %s/%s, код ошибки %d",name,
                EnumToString(period),
                GetLastError());
}

//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Bulls Power
short_name=StringFormat("iBullsPower(%s/%s, period=%d)",name,EnumToString(period),r
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- количество копируемых значений из индикатора iBullsPower
    int values_to_copy;
    //--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
        return(0);
    }

    //--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
    //--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
}

```

```

{
    //--- если массив iBullsPowerBuffer больше, чем значений в индикаторе iBullsPower
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массив iBullsPowerBuffer значениями из индикатора Bulls Power
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(iBullsPowerBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Bulls Power
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iBullsPower |+
//+-----+
bool FillArrayFromBuffer(double &values[], // индикаторный буфер значений Bulls Power
                        int ind_handle, // хэндл индикатора iBullsPower
                        int amount // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iBullsPowerBuffer значениями из индикаторного буфера подряд
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iBullsPower, код ошибки %d", GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считать
    return(false);
}

//--- все получилось
return(true);
}

//+-----+

```

```
//| Indicator deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
//+-----+
```

## iCCI

Возвращает хэндл индикатора Commodity Channel Index. Всего один буфер.

```
int iCCI(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения для вычисления индикатора.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//---------------------------------------------------------------------+
//|                               Demo_iCCI.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iCCI."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"

#property indicator_separate_window
#property indicator_buffers 1
```

```

#property indicator_plots    1
//--- построение iCCI
#property indicator_label1  "iCCI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- горизонтальные уровни в окне индикатора
#property indicator_level1 -100.0
#property indicator_level2 100.0
//+-----+
//| Перечисление способов создания хэндла           |
//+-----+
enum Creation
{
    Call_iCCI,          // использовать iCCI
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iCCI;           // тип функции
input int                ma_period=14;            // период скользящей средней
input ENUM_APPLIED_PRICE applied_price=PRICE_TYPICAL; // тип цены
input string              symbol=" ";             // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT; // таймфрейм

//--- индикаторный буфер
double      iCCIBuffer[];

//--- переменная для хранения хэндла индикатора iCCI
int         handle;
//--- переменная для хранения
string      name=symbol;
//--- имя индикатора на графике
string      short_name;
//--- будем хранить количество значений в индикаторе Commodity Channel Index
int         bars_calculated=0;
//+-----+
//| Custom indicator initialization function          |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iCCIBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
}

```

```

//--- возьмем символ с графика, на котором запущен индикатор
name=_Symbol;
}

//--- создадим хэндл индикатора
if(type==Call_iCCI)
    handle=iCCI(name,period,ma_period,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[2];
    //--- период средней
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- тип цены
    pars[1].type=TYPE_INT;
    pars[1].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_CCI,2,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iCCI для пары %s/%s, код ошибки",
               name,
               EnumToString(period),
               GetLastError());
}
//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор CCI
short_name=StringFormat("iCCI(%s/%s, %d, %s)",name,EnumToString(period),
                         ma_period,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])

```

```

{
//--- количество копируемых значений из индикатора iCCI
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
    return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив iCCIBuffer больше, чем значений в индикаторе iCCI на паре symbol
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}
//--- заполняем массив iCCIBuffer значениями из индикатора Commodity Channel Index
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(iCCIBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Commodity Channel Index
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iCCI | 
//+-----+
bool FillArrayFromBuffer(double &values[], // индикаторный буфер значений Commodity CCI
                        int ind_handle, // хэндл индикатора iCCI
                        int amount // количество копируемых значений
)
{
//--- сбросим код ошибки

```

```
ResetLastError();

//--- заполняем часть массива iCCIBuffer значениями из индикаторного буфера под индексом amount
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iCCI, код ошибки %d",GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться неактивным
    return(false);
}

//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function           |
//+-----+

void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iChaikin

Возвращает хэндл индикатора Chaikin Oscillator. Всего один буфер.

```
int iChaikin(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             fast_ma_period,   // быстрый период
    int             slow_ma_period,   // медленный период
    ENUM_MA_METHOD ma_method,        // тип склаживания
    ENUM_APPLIED_VOLUME applied_volume // используемый объем
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*fast\_ma\_period*

[in] Быстрый период усреднения для вычисления индикатора.

*slow\_ma\_period*

[in] Медленный период усреднения для вычисления индикатора.

*ma\_method*

[in] Тип усреднения. Может быть любой из констант усреднения [ENUM\\_MA\\_METHOD](#).

*applied\_volume*

[in] Используемый объем. Может быть любой из перечисления [ENUM\\_APPLIED\\_VOLUME](#).

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iChaikin.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
```

```

#property description "индикаторных буферов для технического индикатора iChaikin."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- построение iChaikin
#property indicator_label1 "iChaikin"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrLightSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iChaikin,           // использовать iChaikin
    Call_IndicatorCreate      // использовать IndicatorCreate
};

//--- входные параметры
input Creation            type=Call_iChaikin;          // тип функции
input int                  fast_ma_period=3;           // быстрый период
input int                  slow_ma_period=10;          // медленный период
input ENUM_MA_METHOD       ma_method=MODE_EMA;          // тип сглаживания
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // тип объема
input string               symbol=" ";                 // символ
input ENUM_TIMEFRAMES     period=PERIOD_CURRENT;     // таймфрейм
//--- индикаторный буфер
double                    iChaikinBuffer[];

//--- переменная для хранения хэндла индикатора iChaikin
int          handle;

//--- переменная для хранения
string       name=symbol;
//--- имя индикатора на графике
string       short_name;
//--- будем хранить количество значений в индикаторе Chaikin Oscillator
int          bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iChaikinBuffer,INDICATOR_DATA);
    //--- определимся с символом, на котором строится индикатор
    name=symbol;
}

```

```

//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iChaikin)
    handle=iChaikin(name,period,fast_ma_period,slow_ma_period,ma_method,applied_volume);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[4];
    //--- быстрый период
    pars[0].type=TYPE_INT;
    pars[0].integer_value=fast_ma_period;
    //--- медленный период
    pars[1].type=TYPE_INT;
    pars[1].integer_value=slow_ma_period;
    //--- тип склаживания
    pars[2].type=TYPE_INT;
    pars[2].integer_value=ma_method;
    //--- тип объема
    pars[3].type=TYPE_INT;
    pars[3].integer_value=applied_volume;
    handle=IndicatorCreate(name,period,IND_CHAIKIN,4,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iChaikin для пары %s/%s, код ошибки %d",name,
        EnumToString(period),
        GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Chaikin Oscillator
short_name=StringFormat("iChaikin(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
    fast_ma_period,slow_ma_period,
    EnumToString(ma_method),EnumToString(applied_volume));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

```

```

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- количество копируемых значений из индикатора iChaikin
    int values_to_copy;
    //--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }
    //--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
    //--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- если массив iChaikinBuffer больше, чем значений в индикаторе iChaikin на один бар
        //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
        //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
        //--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- заполняем массив iChaikinBuffer значениями из индикатора Chaikin Oscillator
    //--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
    if(!FillArrayFromBuffer(iChaikinBuffer,handle,values_to_copy)) return(0);
    //--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
    //--- выведем на график служебное сообщение
    Comment(comm);
    //--- запомним количество значений в индикаторе Chaikin Oscillator
}

```

```
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iChaikin           |
//+-----+
bool FillArrayFromBuffer(double &values[], // индикаторный буфер значений Chaikin Oscillator
                         int ind_handle, // хэндл индикатора iChaikin
                         int amount       // количество копируемых значений
                        )
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iChaikinBuffer значениями из индикаторного буфера под индикатором
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iChaikin, код ошибки %d", GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться удаленным
return(false);
}

//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function                           |
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iCustom

Возвращает хэндл указанного пользовательского индикатора.

```
int iCustom(
    string      symbol,        // имя символа
    ENUM_TIMEFRAMES period,   // период
    string      name,         // папка/имя_пользовательского индикатора
    ...
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*name*

[in] Имя пользовательского индикатора, содержащее путь относительно корневой директории индикаторов (MQL5/Indicators/). Если индикатор находится в поддиректории, например, в MQL5/Indicators/Examples, то имя должно выглядеть соответственно, а именно - "Examples\Имя\_индикатора" (обязательно указание двойного обратного слеша вместо одиночного в качестве разделителя).

...

[in] [input-параметры](#) пользовательского индикатора, разделенные запятыми. Тип и порядок следования параметров должен соответствовать. Если параметры не указаны, то будут использованы [значения по умолчанию](#).

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

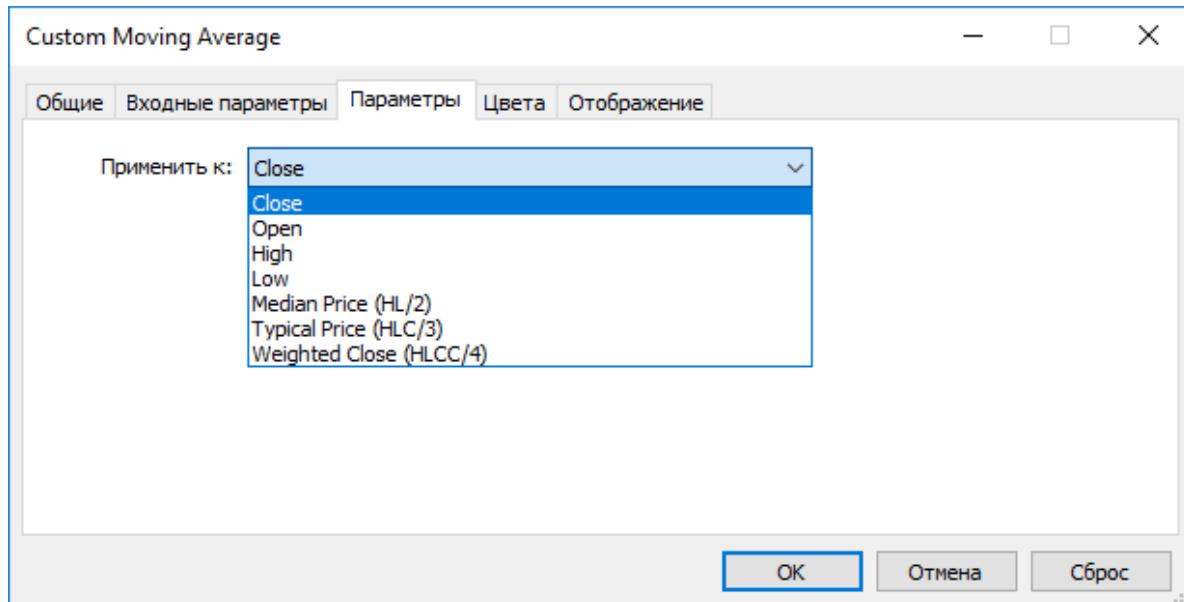
### Примечание

Пользовательский индикатор должен быть скомпилирован (файл с расширением EX5) и находится в директории MQL5/Indicators клиентского терминала или вложенной поддиректории.

Необходимые для тестирования индикаторы определяются автоматически из вызова функций `iCustom()`, если соответствующий параметр задан [константной строкой](#). Для остальных случаев (использование функции [IndicatorCreate\(\)](#) или использование неконстантной строки в параметре, задающем имя индикатора) необходимо указать свойство `#property tester_indicator:`

```
#property tester_indicator "indicator_name.ex5"
```

Если в индикаторе используется [первая форма вызова](#), то при запуске пользовательского индикатора на вкладке "Parameters" можно дополнительно указать на каких данных он будет рассчитываться. Если параметр "Apply to" не выбран явно, то по умолчанию расчет производится по значениям "Close".



При вызове пользовательского индикатора из mq5-программы параметр Applied\_Price или хэндл другого индикатора должен передаваться последним после всех предусмотренных пользовательским индикатором входных переменных.

#### Смотри также

[Свойства программ](#), [Доступ к таймсерием и индикаторам](#), [IndicatorCreate\(\)](#), [IndicatorRelease\(\)](#)

#### Пример:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- plot Label1
#property indicator_label1 "Label1"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- input parameters
input int MA_Period=21;
input int MA_Shift=0;
input ENUM_MA_METHOD MA_Method=MODE_SMA;
//--- indicator buffers
double Label1Buffer[];
//--- хэндл пользовательского индикатора Custom Moving Average.mq5
int MA_handle;
//+-----+
//| Custom indicator initialization function |
```

```

//+-----+
int OnInit()
{
//--- indicator buffers mapping
SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
ResetLastError();
MA_Handle=iCustom(NULL,0,"Examples\\Custom Moving Average",
                  MA_Period,
                  MA_Shift,
                  MA_Method,
                  PRICE_CLOSE // считаем по ценам закрытия
);
Print("MA_Handle = ",MA_Handle," error = ",GetLastError());
//---
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- скопируем значения индикатора Custom Moving Average в наш индикаторный буфер
int copy=CopyBuffer(MA_Handle,0,0,rates_total,Label1Buffer);
Print("copy =",copy," rates_total =",rates_total);
//--- если попытка неудачная - сообщим об этом
if(copy<=0)
    Print("Неудачная попытка получить значения индикатора Custom Moving Average");
//--- return value of prev_calculated for next call
return(rates_total);
}
//+-----+

```

## iDEMA

Возвращает хэндл индикатора Double Exponential Moving Average. Всего один буфер.

```
int iDEMA(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ma_shift,         // смещение индикатора по горизонтали
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления ENUM\_TIMEFRAMES, 0 означает текущий таймфрейм.

*ma\_period*

[in] Период(количество баров) для вычисления индикатора.

*ma\_shift*

[in] Сдвиг индикатора относительно ценового графика.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант ENUM\_APPLIED\_PRICE или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает INVALID\_HANDLE. Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция IndicatorRelease(), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iDEMA.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iDEMA."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
```

```

#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- построение iDEMA
#property indicator_label1 "iDEMA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+

enum Creation
{
    Call_iDEMA,           // использовать iDEMA
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iDEMA;           // тип функции
input int                ma_period=14;            // период скользящей средней
input int                ma_shift=0;             // смещение
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string              symbol=" ";           // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT; // таймфрейм

//--- индикаторный буфер
double                 iDEMABuffer[];

//--- переменная для хранения хэндла индикатора iDEMA
int       handle;
//--- переменная для хранения
string    name=symbol;
//--- имя индикатора на графике
string    short_name;
//--- будем хранить количество значений в индикаторе Double Exponential Moving Average
int       bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iDEMABuffer,INDICATOR_DATA);
//--- зададим смещение
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
}

```

```

        StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iDEMA)
    handle=iDEMA(name,period,ma_period,ma_shift,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[3];
    //--- период средней
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- смещение
    pars[1].type=TYPE_INT;
    pars[1].integer_value=ma_shift;
    //--- тип цены
    pars[2].type=TYPE_INT;
    pars[2].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_DEMA,3,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iDEMA для пары %s/%s, код ошибки %d",name,
                EnumToString(period),
                GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Double Exponential Moving Average
short_name=StringFormat("iDEMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                        ma_period,ma_shift,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
```

```

        const double &open[],
        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {

//--- количество копируемых значений из индикатора iDEMA
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }

//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- если массив iDEMABuffer больше, чем значений в индикаторе iDEMA на паре symbols
        //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
        //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
        //--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }

//--- заполняем массив iDEMABuffer значениями из индикатора Double Exponential Moving Average
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
    if(!FillArrayFromBuffer(iDEMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);

//--- выведем на график служебное сообщение
    Comment(comm);
//--- запомним количество значений в индикаторе Double Exponential Moving Average
    bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
    return(rates_total);
}
//+-----+
// | Заполняем индикаторный буфер из индикатора iDEMA |

```

```
//+-----+
bool FillArrayFromBuffer(double &values[],           // индикаторный буфер значений Double Expon
                         int shift,                  // смещение
                         int ind_handle,             // хэндл индикатора iDEMA
                         int amount)                 // количество копируемых значений
{
//--- сбросим код ошибки
    ResetLastError();
//--- заполняем часть массива iDEMABuffer значениями из индикаторного буфера под индексом shift
    if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
    {
//--- если копирование не удалось, сообщим код ошибки
        PrintFormat("Не удалось скопировать данные из индикатора iDEMA, код ошибки %d",CopyBufferError);
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться удаленным
        return(false);
    }
//--- все получилось
    return(true);
}
//+-----+
//| Indicator deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
    Comment("");
}
```

## iDeMarker

Возвращает хэндл индикатора DeMarker. Всего один буфер.

```
int iDeMarker(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,         // период
    int             ma_period       // период усреднения
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения для вычисления индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iDeMarker.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iDeMarker."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- построение iDeMarker
#property indicator_label1  "iDeMarker"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- горизонтальные уровни в окне индикатора
#property indicator_level1 0.3
#property indicator_level2 0.7
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iDeMarker,           // использовать iDeMarker
    Call_IndicatorCreate      // использовать IndicatorCreate
};

//--- входные параметры
input Creation             type=Call_iDeMarker;          // тип функции
input int                   ma_period=14;                // период скользящей
input string                symbol=" ";                  // символ
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;     // таймфрейм
//--- индикаторный буфер
double                     iDeMarkerBuffer[];

//--- переменная для хранения хэндла индикатора iDeMarker
int                         handle;
//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе DeMarker
int   bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iDeMarkerBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
//--- создадим хэндл индикатора
    if(type==Call_iDeMarker)
        handle=iDeMarker(name,period,ma_period);
}

```

```

else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[1];
    //--- период средней
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    handle=IndicatorCreate(name,period,IND_DEMARKER,1,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iDeMarker для пары %s/%s, код ошибки %d",name,
               EnumToString(period),
               GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор DeMarker
short_name=StringFormat("iDeMarker(%s/%s, period=%d)",name,EnumToString(period),ma_period);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iDeMarker
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
        return(0);
    }
}

```

```

//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений в индикаторе
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив iDeMarkerBuffer больше, чем значений в индикаторе iDeMarker на один элемент
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массив iDeMarkerBuffer значениями из индикатора DeMarker
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(iDeMarkerBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                         TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                         short_name,
                         values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе DeMarker
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iDeMarker |+
//+-----+
bool FillArrayFromBuffer(double &values[], // индикаторный буфер значений DeMarker
                        int ind_handle, // хэндл индикатора iDeMarker
                        int amount // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iDeMarkerBuffer значениями из индикаторного буфера под индексом 0
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iDeMarker, код ошибки %d", GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считать, что
    return(false);
}
//--- все получилось

```

```
    return(true);
}
//+-----+
//| Indicator deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iEnvelopes

Возвращает хэндл индикатора Envelopes.

```
int iEnvelopes(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период для расчета средней линии
    int             ma_shift,         // смещение индикатора по горизонтали
    ENUM_MA_METHOD ma_method,        // тип склаживания
    ENUM_APPLIED_PRICE applied_price, // тип цены или handle
    double          deviation        // отклонение границ от средней линии
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения основной линии индикатора.

*ma\_shift*

[in] Сдвиг индикатора относительно ценового графика.

*ma\_method*

[in] Метод усреднения. Может быть любым из значений значений перечисления [ENUM\\_MA\\_METHOD](#).

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

*deviation*

[in] Отклонение от основной линии в процентах.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Примечание

Номера буферов: 0 - UPPER\_LINE, 1 - LOWER\_LINE.

### Пример:

```
//+-----+
```

```

//|                               Demo_iEnvelopes.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iEnvelopes."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots    2
//--- построение Upper
#property indicator_label1  "Upper"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- построение Lower
#property indicator_label2  "Lower"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iEnvelopes,           // использовать iEnvelopes
    Call_IndicatorCreate        // использовать IndicatorCreate
};

//--- входные параметры
input Creation              type=Call_iEnvelopes;          // тип функции
input int                    ma_period=14;                  // период скользящей
input int                    ma_shift=0;                   // смещение
input ENUM_MA_METHOD         ma_method=MODE_SMA;            // тип сглаживания
input ENUM_APPLIED_PRICE     applied_price=PRICE_CLOSE;    // тип цены
input double                 deviation=0.1;                // отклонение границ от скользя
input string                 symbol=" ";                  // символ
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;       // таймфрейм
//--- индикаторный буфер
double          UpperBuffer[];
double          LowerBuffer[];
//--- переменная для хранения хэндла индикатора iEnvelopes

```

```

int      handle;
//--- переменная для хранения
string  name=symbol;
//--- имя индикатора на графике
string  short_name;
//--- будем хранить количество значений в индикаторе Envelopes
int     bars_calculated=0;
//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
    SetIndexBuffer(0,UpperBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,LowerBuffer,INDICATOR_DATA);
//--- зададим смещение для каждой линии
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
    PlotIndexSetInteger(1,PLOT_SHIFT,ma_shift);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
        //--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
//--- создадим хэндл индикатора
    if(type==Call_iEnvelopes)
        handle=iEnvelopes(name,period,ma_period,ma_shift,ma_method,applied_price,deviat:
    else
    {
        //--- заполним структуру значениями параметров индикатора
        MqlParam pars[5];
        //--- период средней
        pars[0].type=TYPE_INT;
        pars[0].integer_value=ma_period;
        //--- смещение
        pars[1].type=TYPE_INT;
        pars[1].integer_value=ma_shift;
        //--- тип склаживания
        pars[2].type=TYPE_INT;
        pars[2].integer_value=ma_method;
        //--- тип цены
        pars[3].type=TYPE_INT;
        pars[3].integer_value=applied_price;
        //--- тип цены
    }
}

```

```

pars[4].type=TYPE_DOUBLE;
pars[4].double_value=deviation;
handle=IndicatorCreate(name,period,IND_ENVELOPES,5,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
//--- сообщим о неудаче и выведем номер ошибки
PrintFormat("Не удалось создать хэндл индикатора iEnvelopes для пары %s/%s, код
name,
EnumToString(period),
GetLastError());
//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Envelopes
short_name=StringFormat("iEnvelopes(%s/%s, %d, %d, %s,%s, %G)",name,EnumToString(pe
ma_period,ma_shift,EnumToString(ma_method),EnumToString(applied_price),deviation);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iEnvelopes
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
}
}

```

```

//--- если массив UpperBuffer больше, чем значений в индикаторе iEnvelopes на п
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
if(calculated>rates_total) values_to_copy=rates_total;
else values_to_copy=calculated;
}
else
{
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
//--- для расчета добавилось не более одного бара
values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массивы UpperBuffer и LowerBuffer значениями из индикатора Envelopes
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArraysFromBuffers(UpperBuffer,LowerBuffer,ma_shift,handle,values_to_copy))
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
short_name,
values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Envelopes
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буфера из индикатора iEnvelopes |+
//+-----+
bool FillArraysFromBuffers(double &upper_values[], // индикаторный буфер верхней гр
double &lower_values[], // индикаторный буфер нижней гр
int shift, // смещение
int ind_handle, // хэндл индикатора iEnvelopes
int amount // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();
//--- заполняем часть массива UpperBuffer значениями из индикаторного буфера под индексом shift
if(CopyBuffer(ind_handle,0,-shift,amount,upper_values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iEnvelopes, код ошибки %d", GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться неактивным
return(false);
}
//--- заполняем часть массива LowerBuffer значениями из индикаторного буфера под индексом shift
if(CopyBuffer(ind_handle,1,-shift,amount,lower_values)<0)
{
}
}

```

```
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iEnvelopes, код ошибки
//--- завершим с нулевым результатом - это означает, что индикатор будет считат
return(false);
}

//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iForce

Возвращает хэндл индикатора Force Index. Всего один буфер.

```
int iForce(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    ENUM_MA_METHOD ma_method,        // тип сглаживания
    ENUM_APPLIED_VOLUME applied_volume // тип объема для расчета
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления ENUM\_TIMEFRAMES, 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения для вычисления индикатора.

*ma\_method*

[in] Метод усреднения. Может быть любым из значений перечисления ENUM\_MA\_METHOD.

*applied\_volume*

[in] Используемый объем. Может быть любой из значений перечисления ENUM\_APPLIED\_VOLUME.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает INVALID\_HANDLE. Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция IndicatorRelease(), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iForce.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iForce."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
```

```

#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- построение iForce
#property indicator_label1 "iForce"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrLightSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+

enum Creation
{
    Call_iForce,           // использовать iForce
    Call_IndicatorCreate   // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iForce;           // тип функции
input int                ma_period=13;             // период усреднения
input ENUM_MA_METHOD     ma_method=MODE_SMA;         // тип склаживания
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // тип объема
input string              symbol=" ";               // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT;    // таймфрейм

//--- индикаторный буфер
double                  iForceBuffer[];

//--- переменная для хранения хэндла индикатора iForce
int                     handle;
//--- переменная для хранения
string                 name=symbol;
//--- имя индикатора на графике
string                 short_name;
//--- будем хранить количество значений в индикаторе Force
int                     bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
    //--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iForceBuffer,INDICATOR_DATA);
    //--- определимся с символом, на котором строится индикатор
    name=symbol;
    //--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
    //--- если после этого длина строки name нулевая
}

```

```

if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}

//--- создадим хэндл индикатора
if(type==Call_iForce)
    handle=iForce(name,period,ma_period,ma_method,applied_volume);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[3];
    //--- период средней
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- тип склаживания
    pars[1].type=TYPE_INT;
    pars[1].integer_value=ma_method;
    //--- тип объема
    pars[2].type=TYPE_INT;
    pars[2].integer_value=applied_volume;
    //--- тип цены
    handle=IndicatorCreate(name,period,IND_FORCE,3,pars);
}

//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iForce для пары %s/%s, код ошибки %d", name,
               EnumToString(period),
               GetLastError());
}

//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Force
short_name=StringFormat("iForce(%s/%s, %d, %s, %s)",name,EnumToString(period),
                        ma_period,EnumToString(ma_method),EnumToString(applied_volume));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
```

```

        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {

//--- количество копируемых значений из индикатора iForce
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError
        return(0);
    }

//--- если это первый запуск вычислений нашего индикатора или изменилось количество з
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
    {
        //--- если массив iForceBuffer больше, чем значений в индикаторе iForce на паре
        //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else                         values_to_copy=calculated;
    }
    else
    {
        //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
        //--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }

//--- заполняем массив iForceBuffer значениями из индикатора Force
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем рабо
    if(!FillArrayFromBuffer(iForceBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
//--- выведем на график служебное сообщение
    Comment(comm);
//--- запомним количество значений в индикаторе Force
    bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
    return(rates_total);
}

//-----+
//| Заполняем индикаторный буфер из индикатора iForce |
//-----+

```

```
bool FillArrayFromBuffer(double &values[], // индикаторный буфер значений Force Index
                        int ind_handle, // хэндл индикатора iForce
                        int amount       // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();
//--- заполняем часть массива iForceBuffer значениями из индикаторного буфера под индексом
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iForce, код ошибки %d",
//--- завершим с нулевым результатом - это означает, что индикатор будет считать,
return(false);
}
//--- все получилось
return(true);
}
//+-----+
//| Indicator deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iFractals

Возвращает хэндл индикатора Fractals.

```
int iFractals(
    string          symbol,      // имя символа
    ENUM_TIMEFRAMES period      // период
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Примечание

Номера буферов: 0 - UPPER\_LINE, 1 - LOWER\_LINE.

### Пример:

```
//+-----+
//|                               Demo_iFractals.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iFractals."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//--- построение FractalUp
```

```

#property indicator_label1 "FractalUp"
#property indicator_type1 DRAW_ARROW
#property indicator_color1 clrBlue
//--- построение FractalDown
#property indicator_label2 "FractalDown"
#property indicator_type2 DRAW_ARROW
#property indicator_color2 clrRed
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iFractals,           // использовать iFractals
    Call_IndicatorCreate      // использовать IndicatorCreate
};

//--- входные параметры
input Creation             type=Call_iFractals;          // тип функции
input string                symbol=" ";                   // символ
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;        // таймфрейм

//--- индикаторные буферы
double          FractalUpBuffer[];
double          FractalDownBuffer[];

//--- переменная для хранения хэндла индикатора iFractals
int            handle;

//--- переменная для хранения
string         name=symbol;
//--- имя индикатора на графике
string         short_name;
//--- будем хранить количество значений в индикаторе Fractals
int            bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
    SetIndexBuffer(0,FractalUpBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,FractalDownBuffer,INDICATOR_DATA);

//--- зададим коды символом из набора Wingdings для свойств PLOT_ARROW
    PlotIndexSetInteger(0,PLOT_ARROW,217); // стрелка вверх
    PlotIndexSetInteger(1,PLOT_ARROW,218); // стрелка вниз

//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
}

```

```

//--- возьмем символ с графика, на котором запущен индикатор
name=_Symbol;
}

//--- создадим хэндл индикатора
if(type==Call_iFractals)
    handle=iFractals(name,period);
else
    handle=IndicatorCreate(name,period,IND_FRACTALS);
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iFractals для пары %s/%s, код ошибки %d", name,
        EnumToString(period),
        GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Fractals
short_name=StringFormat("iFractals(%s/%s)",name,EnumToString(period));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iFractals
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
        return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з

```

```

//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив FractalUpBuffer больше, чем значений в индикаторе iFractals на
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else                         values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массивы FractalUpBuffer и FractalDownBuffer значениями из индикатора iFractals
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArraysFromBuffers(FractalUpBuffer,FractalDownBuffer,handle,values_to_copy))
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Fractals
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буфера из индикатора iFractals |+
//+-----+
bool FillArraysFromBuffers(double &up_arrows[],           // индикаторный буфер стрелок вверх
                           double &down_arrows[],          // индикаторный буфер стрелок вниз
                           int ind_handle,                // хэндл индикатора iFractals
                           int amount                      // количество копируемых значений
                           )
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива FractalUpBuffer значениями из индикаторного буфера под индексом 0
if(CopyBuffer(ind_handle,0,0,amount,up_arrows)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iFractals в массив FractalUpBuffer. Код ошибки: %d\n",
           GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться недействительным
    return(false);
}

```

```
//--- заполняем часть массива FractalDownBuffer значениями из индикаторного буфера под
if(CopyBuffer(ind_handle,1,0,amount,down_arrows)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iFractals в массив FractalDownBuffer");
    GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться
    return(false);
}
//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iFrAMA

Возвращает хэндл индикатора Fractal Adaptive Moving Average. Всего один буфер.

```
int iFrAMA(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ma_shift,         // смещение индикатора по горизонтали
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период(количество баров) для вычисления индикатора.

*ma\_shift*

[in] Сдвиг индикатора относительно ценового графика.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iFrAMA.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iFrAMA."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
```

```

#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- построение iFrAMA
#property indicator_label1 "iFrAMA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+

enum Creation
{
    Call_iFrAMA,           // использовать iFrAMA
    Call_IndicatorCreate   // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iFrAMA;           // тип функции
input int                 ma_period=14;             // период усреднения
input int                 ma_shift=0;              // смещение
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string               symbol=" ";            // символ
input ENUM_TIMEFRAMES     period=PERIOD_CURRENT; // таймфрейм

//--- индикаторный буфер
double                  iFrAMABuffer[];

//--- переменная для хранения хэндла индикатора iFrAMA
int         handle;
//--- переменная для хранения
string      name=symbol;
//--- имя индикатора на графике
string      short_name;
//--- будем хранить количество значений в индикаторе Fractal Adaptive Moving Average
int         bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iFrAMABuffer,INDICATOR_DATA);
//--- зададим смещение
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
}

```

```

        StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iFrAMA)
    handle=iFrAMA(name,period,ma_period,ma_shift,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[3];
    //--- период средней
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- смещение
    pars[1].type=TYPE_INT;
    pars[1].integer_value=ma_shift;
    //--- тип цены
    pars[2].type=TYPE_INT;
    pars[2].integer_value=applied_price;
    //--- тип цены
    handle=IndicatorCreate(name,period,IND_FRAMA,3,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iFrAMA для пары %s/%s, код ошибки %d", name, EnumToString(period), GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор iFrAMA
short_name=StringFormat("iFrAMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                         ma_period,ma_shift,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,

```

```

        const datetime &time[],
        const double &open[],
        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {

//--- количество копируемых значений из индикатора iFrAMA
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество зв
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
//--- если массив iFrAMABuffer больше, чем значений в индикаторе iFrAMA на паре
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
//--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- заполняем массив iFrAMABuffer значениями из индикатора Fractal Adaptive Moving
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем рабо
    if(!FillArrayFromBuffer(iFrAMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
//--- выведем на график служебное сообщение
    Comment(comm);
//--- запомним количество значений в индикаторе Fractal Adaptive Moving Average
    bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
    return(rates_total);
}
//+-----+

```

```
//| Заполняем индикаторный буфер из индикатора iFrAMA
//+-----+
bool FillArrayFromBuffer(double &values[],           // индикаторный буфер значений Fractal Adapter
                         int shift,                  // смещение
                         int ind_handle,             // хэндл индикатора iFrAMA
                         int amount)                 // количество копируемых значений
{
    //--- сбросим код ошибки
    ResetLastError();
    //--- заполняем часть массива iFrAMABuffer значениями из индикаторного буфера под индексом shift
    if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
    {
        //--- если копирование не удалось, сообщим код ошибки
        PrintFormat("Не удалось скопировать данные из индикатора iFrAMA, код ошибки %d",
                    //--- завершим с нулевым результатом - это означает, что индикатор будет считать,
                    return(false);
    }
    //--- все получилось
    return(true);
}
//+-----+
//| Indicator deinitialization function
//+-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iGator

Возвращает хэндл индикатора Gator. Осциллятор показывает разницу между синей и красной линией Аллигатора (верхняя гистограмма) и разницу между красной и зеленой линией (нижняя гистограмма).

```
int iGator(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             jaw_period,       // период для расчета челюстей
    int             jaw_shift,         // смещение челюстей по горизонтали
    int             teeth_period,      // период для расчета зубов
    int             teeth_shift,       // смещение челюстей по зубов
    int             lips_period,       // период для расчета губ
    int             lips_shift,        // смещение губ по горизонтали
    ENUM_MA_METHOD ma_method,        // тип сглаживания
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*jaw\_period*

[in] Период усреднения синей линии (челюсти аллигатора).

*jaw\_shift*

[in] Смещение синей линии Аллигатора относительно графика цены. Не имеет напрямую отношения к визуальному смещению гистограммы индикатора.

*teeth\_period*

[in] Период усреднения красной линии (зубов аллигатора).

*teeth\_shift*

[in] Смещение красной линии Аллигатора относительно графика цены. Не имеет напрямую отношения к визуальному смещению гистограммы индикатора.

*lips\_period*

[in] Период усреднения зеленой линии (губ аллигатора).

*lips\_shift*

[in] Смещение зеленой линии Аллигатора относительно графика цены. Не имеет напрямую отношения к визуальному смещению гистограммы индикатора.

*ma\_method*

[in] Метод усреднения. Может быть любым из значений [ENUM\\_MA\\_METHOD](#).

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

#### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

#### Примечание

Номера буферов: 0 - UPPER\_HISTOGRAM, 1- цветовой буфер верхней гистограммы, 2 - LOWER\_HISTOGRAM, 3- цветовой буфер нижней гистограммы.

#### Пример:

```
//+-----+
//|                               Demo_iGator.mq5  |
//|           Copyright 2011, MetaQuotes Software Corp.  |
//|                               https://www.mql5.com  |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iGator."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)
#property description "Все остальные параметры как в стандартном Gator Oscillator."

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
//--- построение GatorUp
#property indicator_label1  "GatorUp"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- построение GatorDown
#property indicator_label2  "GatorDown"
#property indicator_type2   DRAW_COLOR_HISTOGRAM
#property indicator_color2  clrGreen, clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+-----+
//| Перечисление способов создания хэндла
//+-----+
enum Creation
```

```

{
  Call_iGator,           // использовать iGator
  Call_IndicatorCreate   // использовать IndicatorCreate
};

//--- входные параметры
© 2000-2019, MetaQuotes Software Corp.
```

```

//--- несмотря на то, что в индикаторе две гистограммы, используется одинаковое смещение
PlotIndexSetInteger(1,PLOT_SHIFT,shift);

//--- определимся с символом, на котором строится индикатор
name=symbol;
//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iGator)
    handle=iGator(name,period,jaw_period,jaw_shift,teeth_period,teeth_shift,
                  lips_period,lips_shift,MA_method,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[8];
    //--- периоды и смещения линий Аллигатора
    pars[0].type=TYPE_INT;
    pars[0].integer_value=jaw_period;
    pars[1].type=TYPE_INT;
    pars[1].integer_value=jaw_shift;
    pars[2].type=TYPE_INT;
    pars[2].integer_value=teeth_period;
    pars[3].type=TYPE_INT;
    pars[3].integer_value=teeth_shift;
    pars[4].type=TYPE_INT;
    pars[4].integer_value=lips_period;
    pars[5].type=TYPE_INT;
    pars[5].integer_value=lips_shift;
    //--- тип склаживания
    pars[6].type=TYPE_INT;
    pars[6].integer_value=MA_method;
    //--- тип цены
    pars[7].type=TYPE_INT;
    pars[7].integer_value=applied_price;
    //--- создадим хэндл
    handle=IndicatorCreate(name,period,IND_GATOR,8,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iGator для пары %s/%s, код ошибки %d", name, symbol, GetLastError());
}

```

```

        name,
        EnumToString(period),
        GetLastError());
    }
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Gator Oscillator
short_name=StringFormat("iGator(%s/%s, %d, %d,%d, %d, %d, %d)",name,EnumToString(jaw_period,jaw_shift,teeth_period,teeth_shift,lips_period,IndicatorSetString(INDICATOR_SHORTNAME,short_name));
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
               const double &close[],
               const long &tick_volume[],
               const long &volume[],
               const int &spread[])
{
//--- количество копируемых значений из индикатора iGator
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
    return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив GatorUpBuffer больше, чем значений в индикаторе iGator на паре
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
    //--- для расчета добавилось не более одного бара
}
}

```

```

        values_to_copy=(rates_total-prev_calculated)+1;
    }

//--- заполняем массивы значениями из индикатора Gator Oscillator
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArraysFromBuffers(GatorUpBuffer,GatorUpColors,GatorDownBuffer,GatorDownColors,
    shift,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
    TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
    short_name,
    values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Gator Oscillator
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буферы из индикатора iGator | |
//+-----+
bool FillArraysFromBuffers(double &ups_buffer[],           // индикаторный буфер для верхней линии
                           double &up_color_buffer[],     // индикаторный буфер для индикаторных цветов
                           double &downs_buffer[],       // индикаторный буфер для нижней линии
                           double &downs_color_buffer[], // индикаторный буфер для индикаторных цветов
                           int u_shift,                  // смещение для верхней и нижней линий
                           int ind_handle,               // хэндл индикатора iGator
                           int amount                    // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();
//--- заполняем часть массива GatorUpBuffer значениями из индикаторного буфера под индикатором iGator
if(CopyBuffer(ind_handle,0,-u_shift,amount,ups_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iGator, код ошибки %d",
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться недействительным
return(false);
}

//--- заполняем часть массива GatorUpColors значениями из индикаторного буфера под индикатором iGator
if(CopyBuffer(ind_handle,1,-u_shift,amount,up_color_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iGator, код ошибки %d",
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться недействительным
return(false);
}
}

```

```
//--- заполняем часть массива GatorDownBuffer значениями из индикаторного буфера под индикатором
if(CopyBuffer(ind_handle,2,-u_shift,amount,downs_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iGator, код ошибки %d",
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться пустым
    return(false);
}

//--- заполняем часть массива GatorDownColors значениями из индикаторного буфера под индикатором
if(CopyBuffer(ind_handle,3,-u_shift,amount,downs_color_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iGator, код ошибки %d",
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться пустым
    return(false);
}

//--- все получилось
return(true);
}

//-----+
//| Indicator deinitialization function
//-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## ilchimoku

Возвращает хэндл индикатора Ichimoku Kinko Hyo.

```
int iIchimoku(
    string      symbol,           // имя символа
    ENUM_TIMEFRAMES period,       // период
    int         tenkan_sen,        // период Tenkan-sen
    int         kijun_sen,         // период Kijun-sen
    int         senkou_span_b     // период Senkou Span B
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления ENUM\_TIMEFRAMES, 0 означает текущий таймфрейм.

*tenkan\_sen*

[in] Период усреднения Tenkan Sen.

*kijun\_sen*

[in] Период усреднения Kijun Sen.

*senkou\_span\_b*

[in] Период усреднения Senkou Span B.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает INVALID\_HANDLE. Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция IndicatorRelease(), которой передается хэндл этого индикатора.

### Примечание

Номера буферов: 0 - TENKANSEN\_LINE, 1 - KIJUNSEN\_LINE, 2 - SENKOUSPANA\_LINE, 3 - SENKOUSPANB\_LINE, 4 - CHIKOUSPAN\_LINE.

### Пример:

```
//+-----+
//|                               Demo_iIchimoku.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
```

```

#property description "индикаторных буферов для технического индикатора iIchimoku."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)
#property description "Все остальные параметры как в стандартном Ichimoku Kinko Hyo."

#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots 4
//--- построение Tenkan_sen
#property indicator_label1 "Tenkan_sen"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- построение Kijun_sen
#property indicator_label2 "Kijun_sen"
#property indicator_type2 DRAW_LINE
#property indicator_color2 clrBlue
#property indicator_style2 STYLE_SOLID
#property indicator_width2 1
//--- построение Senkou_Span
#property indicator_label3 "Senkou Span A;Senkou Span B" // в Data Window будет показано
#property indicator_type3 DRAW_FILLING
#property indicator_color3 clrSandyBrown, clrThistle
#property indicator_style3 STYLE_SOLID
#property indicator_width3 1
//--- построение Chikou_Span
#property indicator_label4 "Chinkou_Span"
#property indicator_type4 DRAW_LINE
#property indicator_color4 clrLime
#property indicator_style4 STYLE_SOLID
#property indicator_width4 1
//+-----+
// | Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iIchimoku,           // использовать iIchimoku
    Call_IndicatorCreate       // использовать IndicatorCreate
};

//--- входные параметры
input Creation             type=Call_iIchimoku;          // тип функции
input int                   tenkan_sen=9;                // период Tenkan-sen
input int                   kijun_sen=26;               // период Kijun-sen
input int                   senkou_span_b=52;            // период Senkou Span B
input string                symbol=" ";                 // символ
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // таймфрейм
//--- индикаторный буфер

```

```

double          Tenkan_sen_Buffer[];
double          Kijun_sen_Buffer[];
double          Senkou_Span_A_Buffer[];
double          Senkou_Span_B_Buffer[];
double          Chinkou_Span_Buffer[];

//--- переменная для хранения хэндла индикатора iIchimoku
int      handle;
//--- переменная для хранения
string  name=symbol;
//--- имя индикатора на графике
string  short_name;
//--- будем хранить количество значений в индикаторе Ichimoku Kinko Hyo
int      bars_calculated=0;

//+-----+
//| Custom indicator initialization function           |
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
    SetIndexBuffer(0,Tenkan_sen_Buffer,INDICATOR_DATA);
    SetIndexBuffer(1,Kijun_sen_Buffer,INDICATOR_DATA);
    SetIndexBuffer(2,Senkou_Span_A_Buffer,INDICATOR_DATA);
    SetIndexBuffer(3,Senkou_Span_B_Buffer,INDICATOR_DATA);
    SetIndexBuffer(4,Chinkou_Span_Buffer,INDICATOR_DATA);

//--- зададим смещения для канала Senkou Span на kijun_sen баров в будущее
    PlotIndexSetInteger(2,PLOT_SHIFT,kijun_sen);

//--- для линии Chikou Span смещение задавать не требуется, так как данные Chikou Span
//--- хранятся в индикаторе iIchimoku уже со смещением
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);

//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }

//--- создадим хэндл индикатора
    if(type==Call_iIchimoku)
        handle=iIchimoku(name,period,tenkan_sen,kijun_sen,senkou_span_b);
    else
    {
//--- заполним структуру значениями параметров индикатора
        MqlParam pars[3];
//--- периоды и смещения линий Аллигатора
        pars[0].type=TYPE_INT;
        pars[0].integer_value=tenkan_sen;
    }
}

```

```

pars[1].type=TYPE_INT;
pars[1].integer_value=kijun_sen;
pars[2].type=TYPE_INT;
pars[2].integer_value=senkou_span_b;
//--- создадим хэндл
handle=IndicatorCreate(name,period,IND_ICHIMOKU,3,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
//--- сообщим о неудаче и выведем номер ошибки
PrintFormat("Не удалось создать хэндл индикатора iIchimoku для пары %s/%s, код ошибки %d",name,
EnumToString(period),
GetLastError());
//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Ichimoku Kinko Hyo
short_name=StringFormat("iIchimoku(%s/%s, %d, %d ,%d)",name,EnumToString(period),
tenkan_sen,kijun_sen,senkou_span_b);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iIchimoku
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з

```

```

//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив Tenkan_sen_Buffer больше, чем значений в индикаторе iIchimoku
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else                         values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массивы значениями из индикатора Ichimoku Kinko Hyo
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArraysFromBuffers(Tenkan_sen_Buffer,Kijun_sen_Buffer,Senkou_Span_A_Buffer,S
    kijun_sen,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Ichimoku Kinko Hyo
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буфера из индикатора iIchimoku |+
//+-----+
bool FillArraysFromBuffers(double &tenkan_sen_buffer[],           // индикаторный буфер линии Tenkan
                           double &kijun_sen_buffer[],          // индикаторный буфер линии Kijun
                           double &senkou_span_A_buffer[],     // индикаторный буфер линии Senkou_Span_A
                           double &senkou_span_B_buffer[],     // индикаторный буфер линии Senkou_Span_B
                           double &chinkou_span_buffer[],      // индикаторный буфер Chinkou_Span
                           int senkou_span_shift,             // смещение линий Senkou_Span относительно Tenkan
                           int ind_handle,                   // хэндл индикатора iIchimoku
                           int amount                        // количество копируемых элементов
                           )
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива Tenkan_sen_Buffer значениями из индикаторного буфера под
if(CopyBuffer(ind_handle,0,0,amount,tenkan_sen_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
}

```

```

PrintFormat("1.Не удалось скопировать данные из индикатора iIchimoku, код ошибки
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться не активным
return(false);
}

//--- заполняем часть массива Kijun_sen_Buffer значениями из индикаторного буфера под
if(CopyBuffer(ind_handle,1,0,amount,kijun_sen_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("2.Не удалось скопировать данные из индикатора iIchimoku, код ошибки
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться не активным
return(false);
}

//--- заполняем часть массива Chinkou_Span_Buffer значениями из индикаторного буфера и
//--- при senkou_span_shift>0 линия смещена в будущее на senkou_span_shift баров
if(CopyBuffer(ind_handle,2,-senkou_span_shift,amount,senkou_span_A_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("3.Не удалось скопировать данные из индикатора iIchimoku, код ошибки
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться не активным
return(false);
}

//--- заполняем часть массива Senkou_Span_A_Buffer значениями из индикаторного буфера и
//--- при senkou_span_shift>0 линия смещена в будущее на senkou_span_shift баров
if(CopyBuffer(ind_handle,3,-senkou_span_shift,amount,senkou_span_B_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("4.Не удалось скопировать данные из индикатора iIchimoku, код ошибки
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться не активным
return(false);
}

//--- заполняем часть массива Senkou_Span_B_Buffer значениями из индикаторного буфера и
//--- при копировании Chikou Span смещение учитывать не требуется, так как данные Chikou
//--- хранятся в индикаторе iIchimoku уже со смещением
if(CopyBuffer(ind_handle,4,0,amount,chinkou_span_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("5.Не удалось скопировать данные из индикатора iIchimoku, код ошибки
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться не активным
return(false);
}

//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function |

```

```
//+-----+
void OnDeinit(const int reason)
{
//--- ПОЧИСТИМ график при удалении индикатора
Comment("");
}
```

## iBWMFI

Возвращает хэндл индикатора Market Facilitation Index. Всего один буфер.

```
int iBWMFI(
    string           symbol,          // имя символа
    ENUM_TIMEFRAMES period,         // период
    ENUM_APPLIED_VOLUME applied_volume // тип объема для расчета
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*applied\_volume*

[in] Используемый объем. Может быть любой из значений перечисления [ENUM\\_APPLIED\\_VOLUME](#).

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iBWMFI.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iBWMFI."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- построение iBWMFI
#property indicator_label1 "iBWMFI"
#property indicator_type1  DRAW_COLOR_HISTOGRAM
```

```

#property indicator_color1 clrLime,clrSaddleBrown,clrBlue,clrPink
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iBWMFI,           // использовать iBWMFI
    Call_IndicatorCreate   // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iBWMFI;           // тип функции
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // тип объема
input string              symbol=" ";                 // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT;     // таймфрейм

//--- индикаторный буфер
double      iBWMFIBuffer[];
double      iBWMFIColors[];

//--- переменная для хранения хэндла индикатора iBWMFI
int        handle;

//--- переменная для хранения
string     name=symbol;
//--- имя индикатора на графике
string     short_name;
//--- будем хранить количество значений в индикаторе Market Facilitation Index Билла Блэка
int        bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
    SetIndexBuffer(0,iBWMFIBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,iBWMFIColors,INDICATOR_COLOR_INDEX);

//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }

//--- создадим хэндл индикатора
    if(type==Call_iBWMFI)
        handle=iBWMFI(name,period,applied_volume);
}

```

```

else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[1];
    //--- тип объема
    pars[0].type=TYPE_INT;
    pars[0].integer_value=applied_volume;
    handle=IndicatorCreate(name,period,IND_BWMFI,1,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iBWMFI для пары %s/%s, код ошибки %d",name,
        EnumToString(period),
        GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Market Facilitation Index
short_name=StringFormat("iBWMFI(%s/%s, %s)",name,EnumToString(period),
    EnumToString(applied_volume));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iBWMFI
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
        return(0);
    }
}

```

```

    }

//--- если это первый запуск вычислений нашего индикатора или изменилось количество звездочек в индикаторе
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив iBWMFIBuffer больше, чем значений в индикаторе iBWMFI на пару баров
    //--- в противном случае копировать будем меньше, чем разница индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массивы значениями из индикатора Market Facilitation Index Билла Вильямса
//--- если FillArraysFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArraysFromBuffers(iBWMFIBuffer,iBWMFIColors,handle,values_to_copy)) return;
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Market Facilitation Index Билла Вильямса
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буфера из индикатора iBWMFI |+
//+-----+
bool FillArraysFromBuffers(double &values[],           // индикаторный буфер значений гистограммы
                           double &colors[],          // индикаторный буфер цветов гистограммы
                           int ind_handle,            // хэндл индикатора iBWMFI
                           int amount                  // количество копируемых значений
                           )
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iBWMFIBuffer значениями из индикаторного буфера под индикатором
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iBWMFI, код ошибки %d",
               GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться недействительным
    return(false);
}

```

```
    }

//--- заполняем часть массива iBWMFIColors значениями из индикаторного буфера под индексом amount
if(CopyBuffer(ind_handle,1,0,amount,colors)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iBWMFI, код ошибки %d",
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться неактивным
    return(false);
}

//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function           |
//+-----+

void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iMomentum

Возвращает хэндл индикатора Momentum. Всего один буфер.

```
int iMomentum(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             mom_period,       // период усреднения
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*mom\_period*

[in] Период(количество баров) для вычисления изменения цены.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//---------------------------------------------------------------------+
//|                               Demo_iMomentum.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iMomentum."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном Momentum."
#property indicator_separate_window
```

```

#property indicator_buffers 1
#property indicator_plots 1
//--- plot iMomentum
#property indicator_label1 "iMomentum"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrDodgerBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
// | Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iMomentum,           // использовать iMomentum
    Call_IndicatorCreate      // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iMomentum;           // тип функции
input int                mom_period=14;                 // период моментума
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string              symbol=" ";                  // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT;       // таймфрейм

//--- индикаторный буфер
double iMomentumBuffer[];

//--- переменная для хранения хэндла индикатора iMomentum
int handle;

//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Momentum
int bars_calculated=0;
//+-----+
// | Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iMomentumBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
}

```

```

    }

//--- создадим хэндл индикатора
if(type==Call_iMomentum)
    handle=iMomentum(name,period,mom_period,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[2];
    //--- период
    pars[0].type=TYPE_INT;
    pars[0].integer_value=mom_period;
    //--- тип цены
    pars[1].type=TYPE_INT;
    pars[1].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_MOMENTUM,2,pars);
}

//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iMomentum для пары %s/%s, код ошибки %d", name,
               EnumToString(period),
               GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Momentum
short_name=StringFormat("iMomentum(%s/%s, %d, %s)",name,EnumToString(period),
                        mom_period, EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- количество копируемых значений из индикатора iMomentum
}

```

```

int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
    return(0);
}

//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
//--- или необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив iMomentumBuffer больше, чем значений в индикаторе iMomentum на один элемент
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массив iMomentumBuffer значениями из индикатора Momentum
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(iMomentumBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Momentum
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iMomentum |
//+-----+
bool FillArrayFromBuffer(double &values[], // индикаторный буфер значений Momentum
                        int ind_handle, // хэндл индикатора iMomentum
                        int amount // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();
//--- заполняем часть массива iMomentumBuffer значениями из индикаторного буфера под индексом

```

```
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iMomentum, код ошибки %d", GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться неактивным
    return(false);
}

//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iMFI

Расчет Money Flow Index.

```
int iMFI(
    string           symbol,          // имя символа
    ENUM_TIMEFRAMES period,         // период
    int              ma_period,       // период усреднения
    ENUM_APPLIED_VOLUME applied_volume // тип объема для расчета
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период(количество баров) для вычисления индикатора.

*applied\_volume*

[in] Используемый объем. Может быть любой из [ENUM\\_APPLIED\\_VOLUME](#).

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iMFI.mq5 |
//|           Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iMFI."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном Money Flow Index."

#property indicator_separate_window
#property indicator_buffers 1
```

```

#property indicator_plots    1
//--- построение iMFI
#property indicator_label1  "iMFI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDodgerBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- горизонтальные уровни в окне индикатора
#property indicator_level1  20
#property indicator_level2  80
//+-----+
//| Перечисление способов создания хэндла           |
//+-----+
enum Creation
{
    Call_iMFI,                      // использовать iMFI
    Call_IndicatorCreate             // использовать IndicatorCreate
};

//--- входные параметры
input Creation                  type=Call_iMFI;          // тип функции
input int                         ma_period=14;           // период
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // тип объема
input string                      symbol=" ";            // символ
input ENUM_TIMEFRAMES period=PERIOD_CURRENT;        // таймфрейм
//--- индикаторный буфер
double      iMFIBuffer[];

//--- переменная для хранения хэндла индикатора iMFI
int       handle;
//--- переменная для хранения
string    name=symbol;
//--- имя индикатора на графике
string    short_name;
//--- будем хранить количество значений в индикаторе Money Flow Index
int       bars_calculated=0;
//+-----+
//| Custom indicator initialization function          |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iMFIBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
}

```

```

//--- возьмем символ с графика, на котором запущен индикатор
name=_Symbol;
}

//--- создадим хэндл индикатора
if(type==Call_iMFI)
    handle=iMFI(name,period,ma_period,applied_volume);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[2];
    //--- период
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- тип объема
    pars[1].type=TYPE_INT;
    pars[1].integer_value=applied_volume;
    handle=IndicatorCreate(name,period,IND_MFI,2,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iMFI для пары %s/%s, код ошибки",
               name,
               EnumToString(period),
               GetLastError());
}
//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Money Flow Index
short_name=StringFormat("iMFI(%s/%s, %d, %s)",name,EnumToString(period),
                         ma_period, EnumToString(applied_volume));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])

```

```

{
//--- количество копируемых значений из индикатора iMFI
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
    return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив iMFIBuffer больше, чем значений в индикаторе iMFI на паре symbol
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}
//--- заполняем массив iMFIBuffer значениями из индикатора Money Flow Index
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(iMFIBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Money Flow Index
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
// | Заполняем индикаторный буфер из индикатора iMFI |
//+-----+
bool FillArrayFromBuffer(double &values[], // индикаторный буфер значений Money Flow
                        int ind_handle, // хэндл индикатора iMFI
                        int amount // количество копируемых значений
)
{
//--- сбросим код ошибки

```

```
ResetLastError();

//--- заполняем часть массива iMFIBuffer значениями из индикаторного буфера под индексом
if(CopyBuffer(ind_handle,0,0,amount,values)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iMFI, код ошибки %d",GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться
    return(false);
}

//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iMA

Возвращает хэндл индикатора скользящего среднего. Всего один буфер.

```
int iMA(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ma_shift,         // смещение индикатора по горизонтали
    ENUM_MA_METHOD ma_method,        // тип стгаживания
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения для вычисления скользящего среднего.

*ma\_shift*

[in] Сдвиг индикатора относительно ценового графика.

*ma\_method*

[in] Метод усреднения. Может быть любым из значений [ENUM\\_MA\\_METHOD](#).

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iMA.mq5 |
//|           Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```

#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iMA."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном Moving Average."

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- построение iMA
#property indicator_label1 "iMA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
// | Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iMA, // использовать iMA
    Call_IndicatorCreate // использовать IndicatorCreate
};
//--- входные параметры
input Creation type=Call_iMA; // тип функции
input int ma_period=10; // период средней
input int ma_shift=0; // смещение
input ENUM_MA_METHOD ma_method=MODE_SMA; // тип сглаживания
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string symbol=" "; // символ
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // таймфрейм
//--- индикаторный буфер
double iMABuffer[];
//--- переменная для хранения хэндла индикатора iMA
int handle;
//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Moving Average
int bars_calculated=0;
//+-----+
// | Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- привязка массива к индикаторному буферу
    SetIndexBuffer(0, iMABuffer, INDICATOR_DATA);
}

```

```

//--- зададим смещение
PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- определимся с символом, на котором строится индикатор
name=symbol;
//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iMA)
    handle=iMA(name,period,ma_period,ma_shift,ma_method,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[4];
    //--- период
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- смещение
    pars[1].type=TYPE_INT;
    pars[1].integer_value=ma_shift;
    //--- тип скаживания
    pars[2].type=TYPE_INT;
    pars[2].integer_value=ma_method;
    //--- тип цены
    pars[3].type=TYPE_INT;
    pars[3].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_MA,4,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iMA для пары %s/%s, код ошибки
                name,
                EnumToString(period),
                GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Moving Average
short_name=StringFormat("iMA(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
                        ma_period, ma_shift,EnumToString(ma_method),EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

```

```

//--- нормальное выполнение инициализации индикатора
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iMA
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
//--- если массив iMABuffer больше, чем значений в индикаторе iMA на паре symbol
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
//--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- заполняем массив iMABuffer значениями из индикатора Moving Average
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
    if(!FillArrayFromBuffer(iMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);

```

```
//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Moving Average
bars_calculated=calculated;

//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iMA |+
//+-----+
bool FillArrayFromBuffer(double &values[],           // индикаторный буфер значений Moving Ave
                         int shift,                  // смещение
                         int ind_handle,             // хэндл индикатора iMA
                         int amount)                 // количество копируемых значений
{
    //--- сбросим код ошибки
    ResetLastError();

    //--- заполняем часть массива iMABuffer значениями из индикаторного буфера под индексом shift
    if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
    {
        //--- если копирование не удалось, сообщим код ошибки
        PrintFormat("Не удалось скопировать данные из индикатора iMA, код ошибки %d",GetLastError());
        //--- завершим с нулевым результатом - это означает, что индикатор будет считаться устаревшим
        return(false);
    }

    //--- все получилось
    return(true);
}

//+-----+
//| Indicator deinitialization function |+
//+-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iOsMA

Возвращает хэндл индикатора Moving Average of Oscillator. Осциллятор OsMA показывает разницу между значениями MACD и его сигнальной линии. Всего один буфер.

```
int iOsMA(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             fast_ema_period,   // период быстрой средней
    int             slow_ema_period,   // период медленной средней
    int             signal_period,    // период усреднения разности
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*fast\_ema\_period*

[in] Период усреднения для вычисления быстрой скользящей средней.

*slow\_ema\_period*

[in] Период усреднения для вычисления медленной скользящей средней.

*signal\_period*

[in] Период усреднения для вычисления сигнальной линии.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Примечание

В некоторых системах этот осциллятор называется гистограммой MACD.

### Пример:

```
//+-----+
//|                               Demo_iOsMA.mq5 |
//| Copyright 2011, MetaQuotes Software Corp. |
//| https://www.mql5.com |
```

```

//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iOsMA."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном Moving Average of Oscillator"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots    1
//--- построение iOsMA
#property indicator_label1  "iOsMA"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+-----+
// | Перечисление способов создания хэндла
//+-----+
enum Creation
{
    Call_iOsMA,           // использовать iOsMA
    Call_IndicatorCreate  // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iOsMA;           // тип функции
input int                fast_ema_period=12;        // период быстрой средней
input int                slow_ema_period=26;        // период медленной средней
input int                signal_period=9;         // период усреднения разности
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string              symbol=" ";             // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT; // таймфрейм
//--- индикаторный буфер
double                  iOsMABuffer[];

//--- переменная для хранения хэндла индикатора iAMA
int        handle;
//--- переменная для хранения
string     name=symbol;
//--- имя индикатора на графике
string     short_name;
//--- будем хранить количество значений в индикаторе Moving Average of Oscillator
int        bars_calculated=0;
//+-----+
// | Custom indicator initialization function
//+-----+

```

```

int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iOsMABuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=_symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
        //--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
//--- создадим хэндл индикатора
    if(type==Call_iOsMA)
        handle=iOsMA(name,period,fast_ema_period,slow_ema_period,signal_period,applied_price);
    else
    {
        //--- заполним структуру значениями параметров индикатора
        MqlParam pars[4];
        //--- быстрый период
        pars[0].type=TYPE_INT;
        pars[0].integer_value=fast_ema_period;
        //--- медленный период
        pars[1].type=TYPE_INT;
        pars[1].integer_value=slow_ema_period;
        //--- период усреднения разницы между быстрой и медленной средними
        pars[2].type=TYPE_INT;
        pars[2].integer_value=signal_period;
        //--- тип цены
        pars[3].type=TYPE_INT;
        pars[3].integer_value=applied_price;
        handle=IndicatorCreate(name,period,IND_OSMA,4,pars);
    }
//--- если не удалось создать хэндл
    if(handle==INVALID_HANDLE)
    {
        //--- сообщим о неудаче и выведем номер ошибки
        PrintFormat("Не удалось создать хэндл индикатора iOsMA для пары %s/%s, код ошибки %d",name,
                    EnumToString(period),
                    GetLastError());
        //--- работа индикатора завершается досрочно
        return(INIT_FAILED);
    }
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Moving Average of Oscillator
    short_name=StringFormat("iOsMA(%s/%s,%d,%d,%d,%s)",name,EnumToString(period),
                            fast_ema_period,slow_ema_period,signal_period,applied_price);
}

```

```

        fast_ema_period,slow_ema_period,signal_period,EnumToString
    IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
    return(INIT_SUCCEEDED);
}
//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iOsMA
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError
        return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
    {
//--- если массив iOsMABuffer больше, чем значений в индикаторе iOsMA на паре sy
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнег
//--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- заполняем массивы значениями из индикатора iOsMA
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем рабо
    if(!FillArrayFromBuffer(iOsMABuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),

```

```
short_name,  
values_to_copy);  
//--- выведем на график служебное сообщение  
Comment(comm);  
//--- запомним количество значений в индикаторе Moving Average of Oscillator  
bars_calculated=calculated;  
//--- вернем значение prev_calculated для следующего вызова  
return(rates_total);  
}  
//+-----+  
//| Заполняем индикаторный буфер из индикатора iOsMA |  
//+-----+  
bool FillArrayFromBuffer(double &ama_buffer[], // индикаторный буфер значений OsMA  
                         int ind_handle,           // хэндл индикатора iOsMA  
                         int amount                // количество копируемых значений  
){  
//--- сбросим код ошибки  
ResetLastError();  
//--- заполняем часть массива iOsMABuffer значениями из индикаторного буфера под индексом  
if(CopyBuffer(ind_handle,0,0,amount,ama_buffer)<0)  
{  
//--- если копирование не удалось, сообщим код ошибки  
PrintFormat("Не удалось скопировать данные из индикатора iOsMA, код ошибки %d",  
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться  
return(false);  
}  
//--- все получилось  
return(true);  
}  
//+-----+  
//| Indicator deinitialization function |  
//+-----+  
void OnDeinit(const int reason)  
{  
//--- почистим график при удалении индикатора  
Comment("");  
}
```

## iMACD

Возвращает хэндл индикатора Moving Averages Convergence/Divergence. В тех системах, где OsMA называют гистограммой MACD, данный индикатор изображается в виде двух линий. В клиентском терминале схождение/расхождение скользящих средних рисуется в виде гистограммы.

```
int iMACD(
    string           symbol,          // имя символа
    ENUM_TIMEFRAMES period,         // период
    int              fast_ema_period, // период быстрой средней
    int              slow_ema_period, // период медленной средней
    int              signal_period,   // период усреднения разности
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления ENUM\_TIMEFRAMES, 0 означает текущий таймфрейм.

*fast\_ema\_period*

[in] Период усреднения для вычисления быстрой скользящей средней.

*slow\_ema\_period*

[in] Период усреднения для вычисления медленной скользящей средней.

*signal\_period*

[in] Период усреднения для вычисления сигнальной линии.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант ENUM\_APPLIED\_PRICE или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает INVALID\_HANDLE. Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция IndicatorRelease(), которой передается хэндл этого индикатора.

### Примечание

Номера буферов: 0 - MAIN\_LINE, 1 - SIGNAL\_LINE.

### Пример:

```
//+-----+
//|                                         Demo_iMACD.mq5 |
//|                                         Copyright 2011, MetaQuotes Software Corp. |
```

```
//| https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iMACD."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном MACD."

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 2
//--- построение MACD
#property indicator_label1 "MACD"
#property indicator_type1 DRAW_HISTOGRAM
#property indicator_color1 clrSilver
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- построение Signal
#property indicator_label2 "Signal"
#property indicator_type2 DRAW_LINE
#property indicator_color2 clrRed
#property indicator_style2 STYLE_DOT
#property indicator_width2 1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iMACD,           // использовать iMACD
    Call_IndicatorCreate  // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iMACD;           // тип функции
input int                fast_ema_period=12;        // период быстрой средней
input int                slow_ema_period=26;        // период медленной средней
input int                signal_period=9;          // период усреднения разности
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string              symbol=" ";             // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT;   // таймфрейм

//--- индикаторные буферы
double      MACDBuffer[];
double      SignalBuffer[];

//--- переменная для хранения хэндла индикатора iMACD
int       handle;

//--- переменная для хранения
```

```
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- Будем хранить количество значений в индикаторе Moving Averages Convergence/Divergence
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |+
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
SetIndexBuffer(0,MACDBuffer,INDICATOR_DATA);
SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
name=symbol;
//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
//--- возьмем символ с графика, на котором запущен индикатор
name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iMACD)
    handle=iMACD(name,period,fast_ema_period,slow_ema_period,signal_period,applied_price);
else
{
//--- заполним структуру значениями параметров индикатора
MqlParam pars[4];
//--- быстрый период
pars[0].type=TYPE_INT;
pars[0].integer_value=fast_ema_period;
//--- медленный период
pars[1].type=TYPE_INT;
pars[1].integer_value=slow_ema_period;
//--- период усреднения разницы между быстрой и медленной средними
pars[2].type=TYPE_INT;
pars[2].integer_value=signal_period;
//--- тип цены
pars[3].type=TYPE_INT;
pars[3].integer_value=applied_price;
handle=IndicatorCreate(name,period,IND_MACD,4,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
//--- сообщим о неудаче и выведем номер ошибки
}
```

```

PrintFormat("Не удалось создать хэндл индикатора iMACD для пары %s/%s, код ошибки %d", name,
           EnumToString(period),
           GetLastError()));

//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Moving Averages Converge/Diverge
short_name=StringFormat("iMACD(%s/%s,%d,%d,%d,%s)",name,EnumToString(period),
                        fast_ema_period,slow_ema_period,signal_period,EnumToString(signal));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//----------------------------------------------------------------------------------------------------------------+
//| Custom indicator iteration function
//----------------------------------------------------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iMACD
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
//--- если массив MACDBuffer больше, чем значений в индикаторе iMACD на паре symbol
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
}
}

```

```

    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массивы значениями из индикатора iMACD
//--- если FillArraysFromBuffers вернула false, значит данные не готовы - завершаем работу
if(!FillArraysFromBuffers(MACDBuffer,SignalBuffer,handle,values_to_copy)) return(0);

//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Moving Averages Convergence/Divergence
bars_calculated=calculated;

//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буферы из индикатора iMACD | 
//+-----+
bool FillArraysFromBuffers(double &macd_buffer[],           // индикаторный буфер значений MACD
                           double &signal_buffer[],        // индикаторный буфер сигнальной линии
                           int ind_handle,                // хэндл индикатора iMACD
                           int amount                      // количество копируемых значений
                           )
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iMACDBuffer значениями из индикаторного буфера под индексом 0
if(CopyBuffer(ind_handle,0,0,amount,macd_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iMACD, код ошибки %d",CopyBufferError);
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться неактивным
return(false);
}

//--- заполняем часть массива SignalBuffer значениями из индикаторного буфера под индексом 1
if(CopyBuffer(ind_handle,1,0,amount,signal_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iMACD, код ошибки %d",CopyBufferError);
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться неактивным
return(false);
}

//--- все получилось
return(true);
}

```

```
//+-----+
//| Indicator deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iOBV

Возвращает хэндл индикатора On Balance Volume. Всего один буфер.

```
int iOBV(
    string           symbol,          // имя символа
    ENUM_TIMEFRAMES period,          // период
    ENUM_APPLIED_VOLUME applied_volume // тип объема для расчета
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*applied\_volume*

[in] Используемый объем. Может быть любой из значений перечисления [ENUM\\_APPLIED\\_VOLUME](#).

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iOBV.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iOBV."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- iOBV
#property indicator_label1 "iOBV"
#property indicator_type1 DRAW_LINE
```

```

#property indicator_color1 clrLightSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
// | Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iOBV, // использовать iOBV
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation type=Call_iOBV; // тип функции
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // тип объема
input string symbol=" "; // символ
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // таймфрейм

//--- индикаторные буферы
double iOBVBuffer[];

//--- переменная для хранения хэндла индикатора iOBV
int handle;

//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе On Balance Volume
int bars_calculated=0;
//+-----+
// | Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iOBVBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
//--- создадим хэндл индикатора
    if(type==Call_iOBV)
        handle=iOBV(name,period,applied_volume);
    else
    {
}
}

```

```

//--- заполним структуру значениями параметров индикатора
MqlParam pars[1];
//--- тип объема
pars[0].type=TYPE_INT;
pars[0].integer_value=applied_volume;
handle=IndicatorCreate(name,period,IND_OBV,1,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
//--- сообщим о неудаче и выведем номер ошибки
PrintFormat("Не удалось создать хэндл индикатора iOBV для пары %s/%s, код ошибки %d", name,
EnumToString(period),
GetLastError());
//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор On Balance Volume
short_name=StringFormat("iOBV(%s/%s, %s)",name,EnumToString(period),
EnumToString(applied_volume));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iOBV
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з

```

```

//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив iOBVBuffer больше, чем значений в индикаторе iOBV на паре символов
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else                         values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массивы значениями из индикатора iOBV
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(iOBVBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе On Balance Volume
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iOBV | 
//+-----+
bool FillArrayFromBuffer(double &obv_buffer[], // индикаторный буфер значений ОВВ
                        int ind_handle,        // хэндл индикатора iOBV
                        int amount            // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iOBVBuffer значениями из индикаторного буфера под индексом
if(CopyBuffer(ind_handle,0,0,amount,obv_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iOBV, код ошибки %d",GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считать
    return(false);
}
//--- все получилось
return(true);
}

```

```
    }

//+-----+
//| Indicator deinitialization function           |
//+-----+

void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iSAR

Возвращает хэндл индикатора Parabolic Stop and Reverse system. Всего один буфер.

```
int iSAR(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,         // период
    double          step,            // шаг изменения цены - коэффициент ускорения
    double          maximum          // максимальный шаг
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*step*

[in] Шаг изменения цены, обычно 0.02.

*maximum*

[in] Максимальный шаг, обычно 0.2.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iSAR.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iSAR."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном Parabolic Stop and Re

#property indicator_chart_window
#property indicator_buffers 1
```

```

#property indicator_plots    1
//--- построение iSAR
#property indicator_label1  "iSAR"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//+-----+
//| Перечисление способов создания хэндла           |
//+-----+
enum Creation
{
    Call_iSAR,                      // использовать iSAR
    Call_IndicatorCreate             // использовать IndicatorCreate
};

//--- входные параметры
input Creation                  type=Call_iSAR;          // тип функции
input double                     step=0.02;              // шаг - коэффициент ускорения
input double                     maximum=0.2;            // максимальное значение шага
input string                      symbol=" ";            // символ
input ENUM_TIMEFRAMES           period=PERIOD_CURRENT; // таймфрейм

//--- индикаторные буферы
double      iSARBuffer[];

//--- переменная для хранения хэндла индикатора iSAR
int         handle;

//--- переменная для хранения
string      name=symbol;
//--- имя индикатора на графике
string      short_name;
//--- будем хранить количество значений в индикаторе Parabolic SAR
int         bars_calculated=0;

//+-----+
//| Custom indicator initialization function        |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iSARBuffer,INDICATOR_DATA);
//--- установим свойству PLOT_ARROW код символа из набора Wingdings для отображения на графике
    PlotIndexSetInteger(0,PLOT_ARROW,159);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
    }
}

```

```

        name=_Symbol;
    }

//--- создадим хэндл индикатора
if(type==Call_iSAR)
    handle=iSAR(name,period,step,maximum);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[2];
    //--- значение шага
    pars[0].type=TYPE_DOUBLE;
    pars[0].double_value=step;
    //--- предельное значение шага, которое может использоваться при расчетах
    pars[1].type=TYPE_DOUBLE;
    pars[1].double_value=maximum;
    handle=IndicatorCreate(name,period,IND_SAR,2,pars);
}

//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iSAR для пары %s/%s, код ошибки
                name,
                EnumToString(period),
                GetLastError());
}

//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Parabolic SAR
short_name=StringFormat("iSAR(%s/%s, %G, %G)",name,EnumToString(period),
                        step,maximum);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{

```

```

//--- количество копируемых значений из индикатора iSAR
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
    return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив iSARBuffer больше, чем значений в индикаторе iSAR на паре symbol
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}
//--- заполняем массивы значениями из индикатора iSAR
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(iSARBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Parabolic SAR
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//-----+
//| Заполняем индикаторный буфер из индикатора iSAR |
//-----+
bool FillArrayFromBuffer(double &sar_buffer[], // индикаторный буфер значений Parabolic SAR
                        int ind_handle, // хэндл индикатора iSAR
                        int amount // количество копируемых значений
                        )
{
//--- сбросим код ошибки
ResetLastError();
}

```

```
//--- заполняем часть массива iSARBuffer значениями из индикаторного буфера под индексом amount
if(CopyBuffer(ind_handle,0,0,amount,sar_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iSAR, код ошибки %d",GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться неактивным
    return(false);
}
//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iRSI

Возвращает хэндл индикатора Relative Strength Index. Всего один буфер.

```
int iRSI(
    string           symbol,          // имя символа
    ENUM_TIMEFRAMES period,          // период
    int              ma_period,       // период усреднения
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения для вычисления индекса.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//---------------------------------------------------------------------+
//| Demo_iRSI.mq5 |  
//| Copyright 2011, MetaQuotes Software Corp. |  
//| https://www.mql5.com |  
//---------------------------------------------------------------------+  
  
#property copyright "Copyright 2011, MetaQuotes Software Corp."  
#property link      "https://www.mql5.com"  
#property version   "1.00"  
#property description "Индикатор демонстрирует как нужно получать данные"  
#property description "индикаторных буферов для технического индикатора iRSI."  
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"  
#property description "задаются параметрами symbol и period."  
#property description "Способ создания хэндла задается параметром 'type' (тип функции)  
#property description "Все остальные параметры как в стандартном Relative Strength Ind  
  
#property indicator_separate_window
```

```

#property indicator_buffers 1
#property indicator_plots 1
//--- построение iRSI
#property indicator_label1 "iRSI"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrDodgerBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- пределы для отображения значений в окне индикатора
#property indicator_maximum 100
#property indicator_minimum 0
//--- горизонтальные уровни в окне индикатора
#property indicator_level1 70.0
#property indicator_level2 30.0
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iRSI,           // использовать iRSI
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iRSI;           // тип функции
input int                ma_period=14;            // период усреднения
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string              symbol=" ";             // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT; // таймфрейм
//--- индикаторный буфер
double iRSIBuffer[];
//--- переменная для хранения хэндла индикатора iRSI
int handle;
//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Relative Strength Index
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iRSIBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
}

```

```

        StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iRSI)
    handle=iRSI(name,period,ma_period,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[2];
    //--- период средней
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- предельное значение шага, которое может использоваться при расчетах
    pars[1].type=TYPE_INT;
    pars[1].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_RSI,2,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iRSI для пары %s/%s, код ошибки
                name,
                EnumToString(period),
                GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Relative Strength Ind
short_name=StringFormat("iRSI(%s/%s, %d, %d)",name,EnumToString(period),
                         ma_period,applied_price);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
               const int prev_calculated,
               const datetime &time[],
               const double &open[],
               const double &high[],
               const double &low[],
```

```

        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {
//--- количество копируемых значений из индикатора iRSI
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError
        return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество з
//--- или необходимо рассчитать индикатор для двух или более баров (значит что-то
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
    {
//--- если массив iRSIBuffer больше, чем значений в индикаторе iRSI на паре symb
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнег
//--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- заполняем массив значениями из индикатора iRSI
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем рабо
    if(!FillArrayFromBuffer(iRSIBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                                TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                                short_name,
                                values_to_copy);
//--- выведем на график служебное сообщение
    Comment(comm);
//--- запомним количество значений в индикаторе Relative Strength Index
    bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
    return(rates_total);
}
//-----+
//| Заполняем индикаторный буфер из индикатора iRSI |
//-----+
bool FillArrayFromBuffer(double &rssi_buffer[], // индикаторный буфер значений Relativ
                        int ind_handle, // хэндл индикатора iRSI

```

```
        int amount           // количество копируемых значений
    )
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iRSIBuffer значениями из индикаторного буфера под индексом
if(CopyBuffer(ind_handle,0,0,amount,rsi_buffer)<0)
{
//--- если копирование не удалось, сообщим код ошибки
PrintFormat("Не удалось скопировать данные из индикатора iRSI, код ошибки %d",GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться удаленным
return(false);
}
//--- все получилось
return(true);
}

//-----+
//| Indicator deinitialization function          |
//-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iRVI

Возвращает хэндл индикатора Relative Vigor Index.

```
int iRVI(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,         // период
    int             ma_period       // период усреднения
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения для вычисления индекса.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Примечание

Номера буферов: 0 - MAIN\_LINE, 1 - SIGNAL\_LINE.

### Пример:

```
//+-----+
//|                               Demo_iRVI.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iRVI."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном Relative Vigor Index.

#property indicator_separate_window
#property indicator_buffers 2
```

```

#property indicator_plots 2
//--- построение RVI
#property indicator_label1 "RVI"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- построение Signal
#property indicator_label2 "Signal"
#property indicator_type2 DRAW_LINE
#property indicator_color2 clrRed
#property indicator_style2 STYLE_SOLID
#property indicator_width2 1
//+-----+
// | Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iRVI,           // использовать iRVI
    Call_IndicatorCreate // использовать IndicatorCreate
};
//--- входные параметры
input Creation          type=Call_iRVI;           // тип функции
input int                ma_period=10;            // период для расчетов
input string              symbol=" ";             // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT; // таймфрейм
//--- индикаторные буферы
double      RVIBuffer[];
double      SignalBuffer[];
//--- переменная для хранения хэндла индикатора iRVI
int        handle;
//--- переменная для хранения
string     name=symbol;
//--- имя индикатора на графике
string     short_name;
//--- будем хранить количество значений в индикаторе Relative Vigor Index
int        bars_calculated=0;
//+-----+
// | Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
    SetIndexBuffer(0,RVIBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
}

```

```

        StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iRVI)
    handle=iRVI(name,period,ma_period);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[1];
    //--- период для расчетов
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    handle=IndicatorCreate(name,period,IND_RVI,1,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iRVI для пары %s/%s, код ошибки
                name,
                EnumToString(period),
                GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Relative Vigor Index
short_name=StringFormat("iRVI(%s/%s, %d, %d)",name,EnumToString(period),ma_period);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])

```

```

{
//--- количество копируемых значений из индикатора iRVI
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
    return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив RVIBuffer больше, чем значений в индикаторе iRVI на паре symbol
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}
//--- заполняем массивы значениями из индикатора iRVI
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(RVIBuffer,SignalBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Relative Vigor Index
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буфера из индикатора iRVI | 
//+-----+
bool FillArrayFromBuffer(double &rvi_buffer[],           // индикаторный буфер значений Relative Vigor Index
                        double &signal_buffer[],      // индикаторный буфер сигнальной линии
                        int ind_handle,              // хэндл индикатора iRVI
                        int amount                   // количество копируемых значений
)
{
}

```

```
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iRVIBuffer значениями из индикаторного буфера под индексом
if(CopyBuffer(ind_handle,0,0,amount,rvi_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iRVI, код ошибки %d",GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться
    return(false);
}

//--- заполняем часть массива SignalBuffer значениями из индикаторного буфера под индексом
if(CopyBuffer(ind_handle,1,0,amount,signal_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iRVI, код ошибки %d",GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться
    return(false);
}

//--- все получилось
return(true);
}

//-----+
//| Indicator deinitialization function
//-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iStdDev

Возвращает хэндл индикатора Standard Deviation. Всего один буфер.

```
int iStdDev(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ma_shift,         // смещение индикатора по горизонтали
    ENUM_MA_METHOD  ma_method,        // тип склаживания
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период усреднения для вычисления индикатора.

*ma\_shift*

[in] Сдвиг индикатора относительно ценового графика.

*ma\_method*

[in] Метод усреднения. Может быть любым из значений [ENUM\\_MA\\_METHOD](#).

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                                         Demo_iStdDev.mq5 |
//|                                         Copyright 2011, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```

#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iStdDev."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном Standard Deviation."

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots 1
//--- построение iStdDev
#property indicator_label1 "iStdDev"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrMediumSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iStdDev,           // использовать iStdDev
    Call_IndicatorCreate    // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iStdDev;           // тип функции
input int                ma_period=20;             // период усреднения
input int                ma_shift=0;              // смещение
input ENUM_MA_METHOD     ma_method=MODE_SMA;        // тип сглаживания
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string              symbol=" ";            // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT;   // таймфрейм

//--- индикаторный буфер
double                  iStdDevBuffer[];

//--- переменная для хранения хэндла индикатора iStdDev
int         handle;

//--- переменная для хранения
string      name=symbol;

//--- имя индикатора на графике
string      short_name;

//--- будем хранить количество значений в индикаторе Standard Deviation
int         bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iStdDevBuffer,INDICATOR_DATA);
}

```

```

//--- зададим смещение
PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- определимся с символом, на котором строится индикатор
name=symbol;
//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iStdDev)
    handle=iStdDev(name,period,ma_period,ma_shift,ma_method,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[4];
    //--- период
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- смещение
    pars[1].type=TYPE_INT;
    pars[1].integer_value=ma_shift;
    //--- тип скаживания
    pars[2].type=TYPE_INT;
    pars[2].integer_value=ma_method;
    //--- тип цены
    pars[3].type=TYPE_INT;
    pars[3].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_STDDEV,4,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iStdDev для пары %s/%s, код ошибки %d",name,
        EnumToString(period),
        GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Standard Deviation
short_name=StringFormat("iStdDev(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
    ma_period,ma_shift,EnumToString(ma_method),EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

```

```

//--- нормальное выполнение инициализации индикатора
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- количество копируемых значений из индикатора iStdDev
    int values_to_copy;
    //--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }
    //--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
    //--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- если массив iStdDevBuffer больше, чем значений в индикаторе iStdDev на пару баров
        //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
        //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
        //--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- заполняем массив значениями из индикатора Standard Deviation
    //--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
    if(!FillArrayFromBuffer(iStdDevBuffer,ma_shift,handle,values_to_copy)) return(0);
    //--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
}

```

```

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Standard Deviation
bars_calculated=calculated;

//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iStdDev           |
//+-----+
bool FillArrayFromBuffer(double &std_buffer[], // индикаторный буфер линии Standard I
                         int std_shift,          // смещение линии Standard Deviation
                         int ind_handle,         // хэндл индикатора iStdDev
                         int amount)             // количество копируемых значений
{
    //--- сбросим код ошибки
    ResetLastError();

    //--- заполняем часть массива iStdDevBuffer значениями из индикаторного буфера под индикатором
    if(CopyBuffer(ind_handle,0,-std_shift,amount,std_buffer)<0)
    {
        //--- если копирование не удалось, сообщим код ошибки
        PrintFormat("Не удалось скопировать данные из индикатора iStdDev, код ошибки %d",
                    GetLastError());
        //--- завершим с нулевым результатом - это означает, что индикатор будет считаться удаленным
        return(false);
    }

    //--- все получилось
    return(true);
}

//+-----+
//| Indicator deinitialization function                           |
//+-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}

```

## iStochastic

Возвращает хэндл индикатора Stochastic Oscillator.

```
int iStochastic(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             Kperiod,          // К-период (количество баров для расчетов)
    int             Dperiod,          // D-период (период первичного сглаживания)
    int             slowing,           // окончательное сглаживание
    ENUM_MA_METHOD ma_method,        // тип сглаживания
    ENUM_STO_PRICE  price_field     // способ расчета стохастика
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*Kperiod*

[in] К-период(количество баров) для вычисления линии %K.

*Dperiod*

[in] Период усреднения для вычисления линии %D.

*slowing*

[in] Значение замедления.

*ma\_method*

[in] Метод усреднения. Может быть любым из значений [ENUM\\_MA\\_METHOD](#).

*price\_field*

[in] Параметр выбора цен для расчета. Может быть одной из величин [ENUM\\_STO\\_PRICE](#).

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Примечание

Номера буферов: 0 - MAIN\_LINE, 1 - SIGNAL\_LINE.

### Пример:

```
//+-----+
//|                               Demo_iStochastic.mq5 |
```

```

//| Copyright 2011, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iStochastic."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном Stochastic Oscillator"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots 2
//--- построение Stochastic
#property indicator_label1 "Stochastic"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrLightSeaGreen
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- построение Signal
#property indicator_label2 "Signal"
#property indicator_type2 DRAW_LINE
#property indicator_color2 clrRed
#property indicator_style2 STYLE_SOLID
#property indicator_width2 1
//--- зададим граничные значения индикатора
#property indicator_minimum 0
#property indicator_maximum 100
//--- горизонтальные уровни в окне индикатора
#property indicator_level1 -100.0
#property indicator_level2 100.0
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iStochastic,           // использовать iStochastic
    Call_IndicatorCreate        // использовать IndicatorCreate
};

//--- входные параметры
input Creation             type=Call_iStochastic;      // тип функции
input int                   Kperiod=5;                  // K-период (количество баров для
                           // расчета)
input int                   Dperiod=3;                 // D-период (период первичного
                           // расчета)
input int                   slowing=3;                // период для окончательного сглаживания
input ENUM_MA_METHOD        ma_method=MODE_SMA;       // тип сглаживания
input ENUM_STO_PRICE         price_field=STO_LOWHIGH; // способ расчета стохастика

```

```

input string symbol=" "; // СИМВОЛ
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // ТАЙМФРЕЙМ

//--- индикаторные буферы
double StochasticBuffer[];
double SignalBuffer[];

//--- переменная для хранения хэндла индикатора iStochastic
int handle;

//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Stochastic Oscillator
int bars_calculated=0;

//+-----+
//| Custom indicator initialization function | 
//+-----+
int OnInit()
{
//--- привязка массивов к индикаторным буферам
SetIndexBuffer(0,StochasticBuffer,INDICATOR_DATA);
SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);

//--- определимся с символом, на котором строится индикатор
name=symbol;

//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);

//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
//--- возьмем символ с графика, на котором запущен индикатор
name=_Symbol;
}

//--- создадим хэндл индикатора
if(type==Call_iStochastic)
handle=iStochastic(name,period,Kperiod,Dperiod,slowing,ma_method,price_field);
else
{
//--- заполним структуру значениями параметров индикатора
MqlParam pars[5];
//--- период К для расчетов
pars[0].type=TYPE_INT;
pars[0].integer_value=Kperiod;
//--- период Д для первичного сглаживания
pars[1].type=TYPE_INT;
pars[1].integer_value=Dperiod;
//--- период К для окончательного сглаживания
pars[2].type=TYPE_INT;
pars[2].integer_value=slowing;
//--- тип сглаживания
}
}

```

```

pars[3].type=TYPE_INT;
pars[3].integer_value=ma_method;
//--- способ расчета стохастика
pars[4].type=TYPE_INT;
pars[4].integer_value=price_field;
handle=IndicatorCreate(name,period,IND_STOCHASTIC,5,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
//--- сообщим о неудаче и выведем номер ошибки
PrintFormat("Не удалось создать хэндл индикатора iStochastic для пары %s/%s, код ошибки %d",name,
EnumToString(period),
GetLastError());
//--- работа индикатора завершается досрочно
return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Stochastic Oscillator
short_name=StringFormat("iStochastic(%s/%s, %d, %d, %d, %s, %s)",name,EnumToString(
Kperiod,Dperiod,slowing,EnumToString(ma_method),EnumToString(price_field));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iStochastic
int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
return(0);
}
//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
values_to_copy=calculated;
}

```

```

//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив StochasticBuffer больше, чем значений в индикаторе iStochastic
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else                         values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массивы значениями из индикатора iStochastic
//--- если FillArraysFromBuffers вернула false, значит данные не готовы - завершаем расчет
if(!FillArraysFromBuffers(StochasticBuffer,SignalBuffer,handle,values_to_copy)) return;
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Stochastic Oscillator
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буфера из индикатора iStochastic |+
//+-----+
bool FillArraysFromBuffers(double &main_buffer[],      // индикаторный буфер значений Stochastic
                           double &signal_buffer[],   // индикаторный буфер сигнальной линии
                           int ind_handle,           // хэндл индикатора iStochastic
                           int amount                // количество копируемых значений
                           )
{
//--- сбросим код ошибки
ResetLastError();
//--- заполняем часть массива StochasticBuffer значениями из индикаторного буфера под индексом
if(CopyBuffer(ind_handle,MAIN_LINE,0,amount,main_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iStochastic, код ошибки %d", GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считать
    return(false);
}

//--- заполняем часть массива SignalBuffer значениями из индикаторного буфера под индексом

```

```
if(CopyBuffer(ind_handle,SIGNAL_LINE,0,amount,signal_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iStochastic, код ошибки
    //--- завершим с нулевым результатом - это означает, что индикатор будет считат
    return(false);
}

//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}
```

## iTEMA

Возвращает хэндл индикатора Triple Exponential Moving Average. Всего один буфер.

```
int iTEMA(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ma_shift,         // смещение индикатора по горизонтали
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период(количество баров) для вычисления индикатора.

*ma\_shift*

[in] Сдвиг индикатора относительно ценового графика.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iTEMA.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iTEMA."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
```

```

#property description "Способ создания хэндла задается параметром 'type' (тип функции)
#property description "Все остальные параметры как в стандартном Triple Exponential Ma

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- построение iTEMA
#property indicator_label1 "iTEMA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iTEMA, // использовать iTEMA
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation type=Call_iTEMA; // тип функции
input int ma_period=14; // период усреднения
input int ma_shift=0; // смещение
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string symbol=" "; // символ
input ENUM_TIMEFRAMES period=PERIOD_CURRENT; // таймфрейм
//--- индикаторный буфер
double iTEMABuffer[];
//--- переменная для хранения хэндла индикатора iTEMA
int handle;
//--- переменная для хранения
string name=symbol;
//--- имя индикатора на графике
string short_name;
//--- будем хранить количество значений в индикаторе Triple Exponential Moving Average
int bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iTEMABuffer,INDICATOR_DATA);
//--- зададим смещение
    PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
}

```

```

StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iTEMA)
    handle=iTEMA(name,period,ma_period,ma_shift,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[3];
    //--- период
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- смещение
    pars[1].type=TYPE_INT;
    pars[1].integer_value=ma_shift;
    //--- тип цены
    pars[2].type=TYPE_INT;
    pars[2].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_TEMA,3,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iTEMA для пары %s/%s, код ошибки %d",name,
               EnumToString(period),
               GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Triple Exponential Moving Average
short_name=StringFormat("iTEMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                        ma_period,ma_shift,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function |
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,

```

```

        const datetime &time[],
        const double &open[],
        const double &high[],
        const double &low[],
        const double &close[],
        const long &tick_volume[],
        const long &volume[],
        const int &spread[])
    {

//--- количество копируемых значений из индикатора iITEMA
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }
//--- если это первый запуск вычислений нашего индикатора или изменилось количество зв
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
//--- если массив iITEMABuffer больше, чем значений в индикаторе iITEMA на паре sy
//--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else
            values_to_copy=calculated;
    }
    else
    {
//--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
//--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }
//--- заполняем массив значениями из индикатора Triple Exponential Moving Average
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем рабо
    if(!FillArrayFromBuffer(iITEMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
//--- выведем на график служебное сообщение
    Comment(comm);
//--- запомним количество значений в индикаторе Triple Exponential Moving Average
    bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
    return(rates_total);
}
//+-----+

```

```

//| Заполняем индикаторный буфер из индикатора iITEMA
//+-----+
bool FillArrayFromBuffer(double &tema_buffer[], // индикаторный буфер значений Triple
                         int t_shift,           // смещение линии
                         int ind_handle,        // хэндл индикатора iITEMA
                         int amount             // количество копируемых значений
                        )
{
//--- сбросим код ошибки
    ResetLastError();
//--- заполняем часть массива iITEMABuffer значениями из индикаторного буфера под индексом t_shift
    if(CopyBuffer(ind_handle,0,-t_shift,amount,tema_buffer)<0)
    {
//--- если копирование не удалось, сообщим код ошибки
        PrintFormat("Не удалось скопировать данные из индикатора iITEMA, код ошибки %d", GetLastError());
//--- завершим с нулевым результатом - это означает, что индикатор будет считаться пустым
        return(false);
    }
//--- все получилось
    return(true);
}
//+-----+
//| Indicator deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
    Comment("");
}

```

## iTriX

Возвращает хэндл индикатора Triple Exponential Moving Averages Oscillator. Всего один буфер.

```
int iTriX(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*ma\_period*

[in] Период(количество баров) для вычисления индикатора.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//---------------------------------------------------------------------+
//|                               Demo_iTriX.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iTriX."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_separate_window
#property indicator_buffers 1
```

```

#property indicator_plots    1
//--- построение iTriX
#property indicator_label1  "iTrix"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//+-----+
//| Перечисление способов создания хэндла |
//+-----+

enum Creation
{
    Call_iTrix,           // использовать iTrix
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iTrix;           // тип функции
input int                ma_period=14;            // период
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string              symbol=" ";             // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT; // таймфрейм

//--- индикаторный буфер
double      iTriXBuffer[];

//--- переменная для хранения хэндла индикатора iTrix
int         handle;

//--- переменная для хранения
string      name=symbol;
//--- имя индикатора на графике
string      short_name;
//--- будем хранить количество значений в индикаторе Triple Exponential Moving Average
int         bars_calculated=0;

//+-----+
//| Custom indicator initialization function |
//+-----+

int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iTriXBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
}

```

```

//--- создадим хэндл индикатора
if(type==Call_iTriX)
    handle=iTriX(name,period,ma_period,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[2];
    //--- период
    pars[0].type=TYPE_INT;
    pars[0].integer_value=ma_period;
    //--- тип цены
    pars[1].type=TYPE_INT;
    pars[1].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_TRIX,2,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iTriX для пары %s/%s, код ошибки: %d", name,
               EnumToString(period),
               GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Triple Exponential Moving Average
short_name=StringFormat("iTriX(%s/%s, %d, %s)",name,EnumToString(period),
                        ma_period,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iTriX
    int values_to_copy;
}

```

```

//--- узнаем количество рассчитанных значений в индикаторе
int calculated=BarsCalculated(handle);
if(calculated<=0)
{
    PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
    return(0);
}

//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив iTriXBuffer больше, чем значений в индикаторе iTriX на паре symbols
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массив значениями из индикатора Triple Exponential Moving Averages Oscillator
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
if(!FillArrayFromBuffer(iTriXBuffer,handle,values_to_copy)) return(0);

//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Triple Exponential Moving Averages Oscillator
bars_calculated=calculated;

//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//-----+
//| Заполняем индикаторный буфер из индикатора iTriX |+
//-----+
bool FillArrayFromBuffer(double &trix_buffer[], // индикаторный буфер значений Triple Exponential Moving Averages Oscillator
                        int ind_handle,           // хэндл индикатора iTriX
                        int amount                // количество копируемых значений
                       )
{
    //--- сбросим код ошибки
    ResetLastError();

    //--- заполняем часть массива iTriXBuffer значениями из индикаторного буфера под индексом 0
    if(CopyBuffer(ind_handle,0,0,amount,trix_buffer)<0)

```

```
{  
    //--- если копирование не удалось, сообщим код ошибки  
    PrintFormat("Не удалось скопировать данные из индикатора iTriX, код ошибки %d",  
    //--- завершим с нулевым результатом - это означает, что индикатор будет считат  
    return(false);  
}  
//--- все получилось  
return(true);  
}  
//+-----+  
//| Indicator deinitialization function |  
//+-----+  
void OnDeinit(const int reason)  
{  
    //--- почистим график при удалении индикатора  
    Comment("");  
}
```

## iWPR

Возвращает хэндл индикатора Larry Williams' Percent Range. Всего один буфер.

```
int iWPR(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,         // период
    int             calc_period      // период усреднения
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*calc\_period*

[in] Период(количество баров) для вычисления индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iWPR.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iWPR."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- построение iWPR
#property indicator_label1  "iWPR"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrCyan
```

```

#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//--- зададим граничные значения индикатора
#property indicator_minimum -100
#property indicator_maximum 0
//--- горизонтальные уровни в окне индикатора
#property indicator_level1 -20.0
#property indicator_level2 -80.0
//+-----+
// | Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iWPR,           // использовать iWPR
    Call_IndicatorCreate // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iWPR;           // тип функции
input int                calc_period=14;          // период
input string              symbol=" ";             // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT;   // таймфрейм
//--- индикаторный буфер
double                  iWPRBuffer[];
//--- переменная для хранения хэндла индикатора iWPR
int                     handle;
//--- переменная для хранения
string                 name=symbol;
//--- имя индикатора на графике
string                 short_name;
//--- будем хранить количество значений в индикаторе Larry Williams' Percent Range
int                     bars_calculated=0;
//+-----+
// | Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iWPRBuffer,INDICATOR_DATA);
//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }
}

```

```

//--- создадим хэндл индикатора
if(type==Call_iWPR)
    handle=iWPR(name,period,calc_period);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[1];
    //--- период
    pars[0].type=TYPE_INT;
    pars[0].integer_value=calc_period;
    handle=IndicatorCreate(name,period,IND_WPR,1,pars);
}

//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iWPR для пары %s/%s, код ошибки %d", name,
        EnumToString(period),
        GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}

//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Williams' Percent Range
short_name=StringFormat("iWPR(%s/%s, %d)",name,EnumToString(period),calc_period);
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- количество копируемых значений из индикатора iWPR
    int values_to_copy;
    //--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
}

```

```

PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError
return(0);

}

//--- если это первый запуск вычислений нашего индикатора или изменилось количество зв
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated
{
    //--- если массив iWPRBuffer больше, чем значений в индикаторе iWPR на паре symk
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else                         values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массив значениями из индикатора Williams' Percent Range
//--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем рабо
if(!FillArrayFromBuffer(iWPRBuffer,handle,values_to_copy)) return(0);
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                           TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                           short_name,
                           values_to_copy);

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Williams' Percent Range
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iWPR |+
//+-----+
bool FillArrayFromBuffer(double &wpr_buffer[], // индикаторный буфер значений Williams'
                        int ind_handle,           // хэндл индикатора iWPR
                        int amount                // количество копируемых значений
)
{
//--- сбросим код ошибки
ResetLastError();

//--- заполняем часть массива iWPRBuffer значениями из индикаторного буфера под индексом
if(CopyBuffer(ind_handle,0,0,amount,wpr_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iWPR, код ошибки %d",GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считат
}

```

```
    return(false);
}
//--- все получилось
return(true);
}
//+-----+
//| Indicator deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## iVIDyA

Возвращает хэндл индикатора Variable Index Dynamic Average. Всего один буфер.

```
int iVIDyA(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    int             cmo_period,       // период Chande Momentum
    int             ema_period,       // период фактора сглаживания
    int             ma_shift,         // смещение индикатора по горизонтали
    ENUM_APPLIED_PRICE applied_price // тип цены или handle
);
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. [NULL](#) означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*cmo\_period*

[in] Период(количество баров) для вычисления Chande Momentum Oscillator.

*ema\_period*

[in] Период(количество баров) ЕМА для вычисления фактора сглаживания.

*ma\_shift*

[in] Сдвиг индикатора относительно ценового графика.

*applied\_price*

[in] Используемая цена. Может быть любой из ценовых констант [ENUM\\_APPLIED\\_PRICE](#) или хендлом другого индикатора.

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                                         Demo_iVIDyA.mq5 |
//|                                         Copyright 2011, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```

#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iVIDyA."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)"
#property description "Все остальные параметры как в стандартном Variable Index Dynamic"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots 1
//--- построение iVIDyA
#property indicator_label1 "iVIDyA"
#property indicator_type1 DRAW_LINE
#property indicator_color1 clrBlue
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iVIDyA,           // использовать iVIDyA
    Call_IndicatorCreate   // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iVIDyA;           // тип функции
input int                 cmo_period=15;            // период Chande Momentum
input int                 ema_period=12;            // период фактора слаживания
input int                 ma_shift=0;              // смещение
input ENUM_APPLIED_PRICE applied_price=PRICE_CLOSE; // тип цены
input string               symbol=" ";             // символ
input ENUM_TIMEFRAMES     period=PERIOD_CURRENT; // таймфрейм

//--- индикаторный буфер
double                  iVIDyABuffer[];

//--- переменная для хранения хэндла индикатора iVIDyA
int                     handle;
//--- переменная для хранения
string                 name=symbol;
//--- имя индикатора на графике
string                 short_name;
//--- будем хранить количество значений в индикаторе Variable Index Dynamic Average
int                     bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
    //--- привязка массива к индикаторному буферу
    SetIndexBuffer(0,iVIDyABuffer,INDICATOR_DATA);
}

```

```

//--- зададим смещение
PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- определимся с символом, на котором строится индикатор
name=symbol;
//--- удалим пробелы слева и справа
StringTrimRight(name);
StringTrimLeft(name);
//--- если после этого длина строки name нулевая
if(StringLen(name)==0)
{
    //--- возьмем символ с графика, на котором запущен индикатор
    name=_Symbol;
}
//--- создадим хэндл индикатора
if(type==Call_iVIDyA)
    handle=iVIDyA(name,period,cmo_period,ema_period,ma_shift,applied_price);
else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[4];
    //--- период Chande Momentum
    pars[0].type=TYPE_INT;
    pars[0].integer_value=cmo_period;
    //--- период фактора сглаживания
    pars[1].type=TYPE_INT;
    pars[1].integer_value=ema_period;
    //--- смещение
    pars[2].type=TYPE_INT;
    pars[2].integer_value=ma_shift;
    //--- тип цены
    pars[3].type=TYPE_INT;
    pars[3].integer_value=applied_price;
    handle=IndicatorCreate(name,period,IND_VIDYA,4,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iVIDyA для пары %s/%s, код ошибки %d",name,
        EnumToString(period),
        GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Variable Index Dynamically
short_name=StringFormat("iVIDyA(%s/%s, %d, %d, %d, %s)",name,EnumToString(period),
    cmo_period,ema_period,ma_shift,EnumToString(applied_price));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);

```

```

//--- нормальное выполнение инициализации индикатора
    return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+

int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
    //--- количество копируемых значений из индикатора iVIDyA
    int values_to_copy;
    //--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError);
        return(0);
    }
    //--- если это первый запуск вычислений нашего индикатора или изменилось количество значений
    //--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то)
    if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
    {
        //--- если массив iVIDyABuffer больше, чем значений в индикаторе iVIDyA на пару
        //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
        if(calculated>rates_total) values_to_copy=rates_total;
        else values_to_copy=calculated;
    }
    else
    {
        //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего
        //--- для расчета добавилось не более одного бара
        values_to_copy=(rates_total-prev_calculated)+1;
    }
    //--- заполняем массив значениями из индикатора Variable Index Dynamic Average
    //--- если FillArrayFromBuffer вернула false, значит данные не готовы - завершаем работу
    if(!FillArrayFromBuffer(iVIDyABuffer,ma_shift,handle,values_to_copy)) return(0);
    //--- сформируем сообщение
    string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                             TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                             short_name,
                             values_to_copy);
}

```

```

//--- выведем на график служебное сообщение
Comment(comm);

//--- запомним количество значений в индикаторе Variable Index Dynamic Average
bars_calculated=calculated;

//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторный буфер из индикатора iVIDyA           |
//+-----+
bool FillArrayFromBuffer(double &vidya_buffer[], // индикаторный буфер значений Variable Index Dynamic Average
                         int v_shift,           // смещение линии
                         int ind_handle,        // хэндл индикатора iVIDyA
                         int amount)            // количество копируемых значений
{
    //--- сбросим код ошибки
    ResetLastError();

    //--- заполняем часть массива iVIDyABuffer значениями из индикаторного буфера под индексом v_shift
    if(CopyBuffer(ind_handle,0,-v_shift,amount,vidya_buffer)<0)
    {
        //--- если копирование не удалось, сообщим код ошибки
        PrintFormat("Не удалось скопировать данные из индикатора iVIDyA, код ошибки %d",
                    GetLastError());
        //--- завершим с нулевым результатом - это означает, что индикатор будет считаться устаревшим
        return(false);
    }

    //--- все получилось
    return(true);
}

//+-----+
//| Indicator deinitialization function                           |
//+-----+
void OnDeinit(const int reason)
{
    //--- почистим график при удалении индикатора
    Comment("");
}

```

## iVolumes

Возвращает хэндл индикатора, отображающего объемы . Всего один буфер.

```
int iVolumes(
    string          symbol,           // имя символа
    ENUM_TIMEFRAMES period,          // период
    ENUM_APPLIED_VOLUME applied_volume // тип объема
)
```

### Параметры

*symbol*

[in] Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.

*period*

[in] Значение периода может быть одним из значений перечисления [ENUM\\_TIMEFRAMES](#), 0 означает текущий таймфрейм.

*applied\_volume*

[in] Используемый объем. Может быть любой из значений перечисления [ENUM\\_APPLIED\\_VOLUME](#).

### Возвращаемое значение

Возвращает хэндл указанного технического индикатора, в случае неудачи возвращает [INVALID\\_HANDLE](#). Для освобождения памяти компьютера от неиспользуемого больше индикатора служит функция [IndicatorRelease\(\)](#), которой передается хэндл этого индикатора.

### Пример:

```
//+-----+
//|                               Demo_iVolumes.mq5 |
//|                               Copyright 2011, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Индикатор демонстрирует как нужно получать данные"
#property description "индикаторных буферов для технического индикатора iVolumes."
#property description "Символ и таймфрейм, на котором рассчитывается индикатор,"
#property description "задаются параметрами symbol и period."
#property description "Способ создания хэндла задается параметром 'type' (тип функции)

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- построение iVolumes
#property indicator_label1 "iVolumes"
#property indicator_type1  DRAW_COLOR_HISTOGRAM
```

```

#property indicator_color1 clrGreen, clrRed
#property indicator_style1 STYLE_SOLID
#property indicator_width1 1
//+-----+
//| Перечисление способов создания хэндла |
//+-----+
enum Creation
{
    Call_iVolumes,           // использовать iVolumes
    Call_IndicatorCreate    // использовать IndicatorCreate
};

//--- входные параметры
input Creation          type=Call_iVolumes;           // тип функции
input ENUM_APPLIED_VOLUME applied_volume=VOLUME_TICK; // тип объема
input string              symbol=" ";                  // символ
input ENUM_TIMEFRAMES    period=PERIOD_CURRENT;      // таймфрейм

//--- индикаторные буферы
double        iVolumesBuffer[];
double        iVolumesColors[];

//--- переменная для хранения хэндла индикатора iVolumes
int         handle;

//--- переменная для хранения
string       name=symbol;
//--- имя индикатора на графике
string       short_name;
//--- будем хранить количество значений в индикаторе Volumes
int         bars_calculated=0;
//+-----+
//| Custom indicator initialization function |
//+-----+
int OnInit()
{
//--- привязка массива к индикаторным буферам
    SetIndexBuffer(0,iVolumesBuffer,INDICATOR_DATA);
    SetIndexBuffer(1,iVolumesColors,INDICATOR_COLOR_INDEX);

//--- определимся с символом, на котором строится индикатор
    name=symbol;
//--- удалим пробелы слева и справа
    StringTrimRight(name);
    StringTrimLeft(name);
//--- если после этого длина строки name нулевая
    if(StringLen(name)==0)
    {
//--- возьмем символ с графика, на котором запущен индикатор
        name=_Symbol;
    }

//--- создадим хэндл индикатора
    if(type==Call_iVolumes)
        handle=iVolumes(name,period,applied_volume);
}

```

```

else
{
    //--- заполним структуру значениями параметров индикатора
    MqlParam pars[1];
    //--- тип цены
    pars[0].type=TYPE_INT;
    pars[0].integer_value=applied_volume;
    handle=IndicatorCreate(name,period,IND_VOLUMES,1,pars);
}
//--- если не удалось создать хэндл
if(handle==INVALID_HANDLE)
{
    //--- сообщим о неудаче и выведем номер ошибки
    PrintFormat("Не удалось создать хэндл индикатора iVolumes для пары %s/%s, код ошибки %d", name,
        EnumToString(period),
        GetLastError());
    //--- работа индикатора завершается досрочно
    return(INIT_FAILED);
}
//--- покажем на какой паре символ/таймфрейм рассчитан индикатор Volumes
short_name=StringFormat("iVolumes (%s/%s, %s)",name,EnumToString(period),EnumToString(indicator));
IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- нормальное выполнение инициализации индикатора
return(INIT_SUCCEEDED);
}

//+-----+
//| Custom indicator iteration function
//+-----+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
{
//--- количество копируемых значений из индикатора iVolumes
    int values_to_copy;
//--- узнаем количество рассчитанных значений в индикаторе
    int calculated=BarsCalculated(handle);
    if(calculated<=0)
    {
        PrintFormat("BarsCalculated() вернул %d, код ошибки %d",calculated,GetLastError());
        return(0);
    }
}

```

```

//--- если это первый запуск вычислений нашего индикатора или изменилось количество значений в индикаторе iVolumes
//--- или если необходимо рассчитать индикатор для двух или более баров (значит что-то изменилось)
if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculated)
{
    //--- если массив iVolumesBuffer больше, чем значений в индикаторе iVolumes на один элемент
    //--- в противном случае копировать будем меньше, чем размер индикаторных буферов
    if(calculated>rates_total) values_to_copy=rates_total;
    else values_to_copy=calculated;
}
else
{
    //--- значит наш индикатор рассчитывается не в первый раз и с момента последнего расчета
    //--- для расчета добавилось не более одного бара
    values_to_copy=(rates_total-prev_calculated)+1;
}

//--- заполняем массивы значениями из индикатора iVolumes
//--- если FillArraysFromBuffers вернула false, значит данные не готовы - завершаем расчет
if(!FillArraysFromBuffers(iVolumesBuffer,iVolumesColors,handle,values_to_copy)) return;
//--- сформируем сообщение
string comm=StringFormat("%s ==> Обновлено значений в индикаторе %s: %d",
                         TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                         short_name,
                         values_to_copy);
//--- выведем на график служебное сообщение
Comment(comm);
//--- запомним количество значений в индикаторе Volumes
bars_calculated=calculated;
//--- вернем значение prev_calculated для следующего вызова
return(rates_total);
}

//+-----+
//| Заполняем индикаторные буфера из индикатора iVolumes |+
//+-----+
bool FillArraysFromBuffers(double &volume_buffer[],           // индикаторный буфер значений Volumes
                           double &color_buffer[],          // индикаторный буфер цветов
                           int ind_handle,                 // хэндл индикатора iVolumes
                           int amount                      // количество копируемых значений
                           )
{
//--- сбросим код ошибки
ResetLastError();
//--- заполняем часть массива iVolumesBuffer значениями из индикаторного буфера под индикатором
if(CopyBuffer(ind_handle,0,0,amount,volume_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iVolumes, код ошибки %d", GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться недействительным
    return(false);
}

```

```
//--- заполняем часть массива iVolumesColors значениями из индикаторного буфера под индикатором
if(CopyBuffer(ind_handle,1,0,amount,color_buffer)<0)
{
    //--- если копирование не удалось, сообщим код ошибки
    PrintFormat("Не удалось скопировать данные из индикатора iVolumes, код ошибки %d", GetLastError());
    //--- завершим с нулевым результатом - это означает, что индикатор будет считаться неактивным
    return(false);
}
//--- все получилось
return(true);
}

//+-----+
//| Indicator deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
//--- почистим график при удалении индикатора
Comment("");
}
```

## Работа с результатами оптимизации

Функции для организации собственной обработки результатов оптимизации в тестере стратегий. Могут вызываться при оптимизации в агентах тестирования, а также локально в экспертах и скриптах.

При запуске эксперта в тестере стратегий можно создать собственный массив данных на основе простых типов или [простых структур](#) (не содержат строки, объекты класса или объекты динамических массивов). Этот набор данных можно сохранить с помощью функции [FrameAdd\(\)](#) в специальной структуре, называемой фреймом (кадр). Каждый агент при оптимизации эксперта может посыпать в терминал серию фреймов. Все полученные фреймы в порядке поступления от агентов записываются в \*.MQD-файл по имени эксперта в папку каталог\_терминала\МQL5\Files\Tester. Поступление фрейма в клиентский терминал от агента тестирования генерирует событие [TesterPass](#).

Фреймы можно сохранять как в память компьютера, так и в файл с указанным именем. Нет никаких ограничений на количество фреймов со стороны языка MQL5.

Функция	Действие
<a href="#">FrameFirst</a>	Переводит указатель чтения фреймов в начало и сбрасывает ранее установленный фильтр
<a href="#">FrameFilter</a>	Устанавливает фильтр чтения фреймов и переводит указатель на начало
<a href="#">FrameNext</a>	Читает фрейм и перемещает указатель на следующий
<a href="#">FrameInputs</a>	Получает <a href="#">input-параметры</a> , на которых сформирован фрейм
<a href="#">FrameAdd</a>	Добавляет фрейм с данными
<a href="#">ParameterGetRange</a>	Получает для <a href="#">input-переменной</a> информацию о диапазоне значений и шаге изменения при оптимизации эксперта в тестере стратегий
<a href="#">ParameterSetRange</a>	Устанавливает правила использования <a href="#">input-переменной</a> при оптимизации эксперта в тестере стратегий: значение, шаг изменения, начальное и конечное значения

### Смотри также

[Статистика тестирования](#), [Информация о запущенной MQL5-программе](#)

## FrameFirst

Переводит указатель чтения фреймов оптимизации в начало и сбрасывает установленный фильтр.

```
bool FrameFirst();
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

## FrameFilter

Устанавливает фильтр чтения фреймов и переводит указатель на начало.

```
bool FrameFilter(
    const string name,           // публичное имя/метка
    long      id                // публичный id
);
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Если в качестве первого параметра передана пустая строка, то фильтр будет работать только по числовому параметру, то есть будут просматриваться все фреймы с указанным id. Если значение второго параметра равно [ULONG\\_MAX](#), то работает только текстовый фильтр.

Вызов [FrameFilter\("", ULONG\\_MAX\)](#) эквивалентен вызову [FrameFirst\(\)](#), то есть равнозначен отсутствию фильтра.

## FrameNext

Читает текущий фрейм и перемещает указатель на следующий. Существует 2 варианта функции.

### 1. Вызов с получением одного числового значения

```
bool FrameNext(
    ulong& pass,           // номер прохода в оптимизации, на котором добавлен фрейм
    string& name,          // публичное имя/метка
    long& id,              // публичный id
    double& value           // значение
);
```

### 2. Вызов с получением всех данных фрейма

```
bool FrameNext(
    ulong& pass,           // номер прохода в оптимизации, на котором добавлен фрейм
    string& name,          // публичное имя/метка
    long& id,              // публичный id
    double& value,          // значение
    void& data[]            // массив любого типа
);
```

#### Параметры

*pass*

[out] Номер прохода при оптимизации в тестере стратегий.

*name*

[out] Имя идентификатора.

*id*

[out] Значение идентификатора.

*value*

[out] Одиночное числовое значение.

*data*

[out] Массив любого типа.

#### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

#### Примечание

При использовании второго варианта вызова необходимо правильно обрабатывать полученные данные в массиве *data[]*.

## FrameInputs

Получает [input-параметры](#), на которых сформирован фрейм с заданным номером прохода.

```
bool FrameInputs(
    ulong    pass,           // номер прохода в оптимизации
    string&  parameters[],   // массив строк вида "parameterN=valueN"
    uint&    parameters_count // общее количество параметров
);
```

### Параметры

*pass*

[in] Номер прохода при оптимизации в тестере стратегий.

*parameters*

[out] Строковый массив с описанием имен и значений параметров

*parameters\_count*

[out] Количество элементов в массиве *parameters[]*.

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Получив количество строк *parameters\_count* в массиве *parameters[]*, можно организовать цикл для перебора всех записей. Это позволит узнать для заданного номера прохода значения входных параметров эксперта.

## FrameAdd

Добавляет фрейм с данными. Существует 2 варианта функции.

### 1. Добавление данных из файла

```
bool FrameAdd(
    const string name,           // публичное имя/метка
    long id,                     // публичный id
    double value,                // значение
    const string filename        // имя файла с данными
);
```

### 2. Добавление данных из массива любого типа

```
bool FrameAdd(
    const string name,           // публичное имя/метка
    long id,                     // публичный id
    double value,                // значение
    const void& data[]          // массив любого типа
);
```

#### Параметры

*name*

[in] Публичная метка фрейм. Может использоваться для фильтра в функции [FrameFilter\(\)](#).

*id*

[in] Публичный идентификатор фрейма. Может использоваться для фильтра в функции [FrameFilter\(\)](#).

*value*

[in] Числовое значение для записи во фрейм. Предназначен для передачи одиночного результата прохода как в функции [OnTester\(\)](#).

*filename*

[in] Имя файла, в котором находятся данные для добавления во фрейм. Файл должен находиться в папке MQL5/Files.

*data*

[in] Массив любого типа для записи во фрейм. Передается по ссылке.

#### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

## ParameterGetRange

Получает для [input-переменной](#) информацию о диапазоне значений и шаге изменения при оптимизации эксперта в тестере стратегий. Существует 2 варианта функции.

### 1. Получение информации для input-параметра целого типа

```
bool ParameterGetRange(
    const string name,           // имя параметра (input-переменной)
    bool& enable,               // разрешена оптимизация параметра
    long& value,                // значение параметра
    long& start,                // начальное значение
    long& step,                 // шаг изменения
    long& stop                  // конечное значение
);
```

### 2. Получение информации для input-параметра вещественного типа

```
bool ParameterGetRange(
    const string name,           // имя параметра (input-переменной)
    bool& enable,               // разрешена оптимизация параметра
    double& value,               // значение параметра
    double& start,                // начальное значение
    double& step,                 // шаг изменения
    double& stop                  // конечное значение
);
```

#### Параметры

*name*

[in] Идентификатор [input-переменной](#). Такие переменные являются внешними параметрами программы, значения которых можно задавать при запуске на графике или при одиночном тестировании.

*enable*

[out] Признак того, что данный параметр можно использовать для перебора значений в процессе оптимизации в тестере стратегий.

*value*

[out] Значение параметра.

*start*

[out] Начальное значение параметра при оптимизации.

*step*

[out] Шаг изменения параметра при переборе его значений.

*stop*

[out] Конечное значение параметра при оптимизации.

#### Возвращаемое значение

Возвращает true в случае успешного выполнения, иначе false. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

### Примечание

Функция может вызываться только из обработчиков [OnTesterInit\(\)](#), [OnTesterPass\(\)](#) и [OnTesterDeinit\(\)](#). Предназначена для получения значения и диапазона изменения входных параметров эксперта в процессе оптимизации в тестере стратегий.

При вызове в OnTesterInit() полученную информацию можно использовать для переопределения правила перебора любой [input-переменной](#) с помощью функции [ParameterSetRange\(\)](#). Таким образом можно задавать новые значения Start, Stop, Step и даже полностью исключать данный параметр из оптимизации несмотря на настройки в тестере стратегий. Это позволяет создавать собственные сценарии для управления пространством входных параметров при оптимизации, то есть исключать из оптимизации одни параметры в зависимости от значений ключевых параметров эксперта.

### Пример:

```
#property description "Эксперт для демонстрации функции ParameterGetRange()."
#property description "Необходимо запускать в тестере стратегий в режиме оптимизации"
//--- input parameters
input int           Input1=1;
input double        Input2=2.0;
input bool          Input3=false;
input ENUM_DAY_OF_WEEK Input4=SUNDAY;

//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- эксперт предназначен только для работы в тестере стратегий
if(!MQL5InfoInteger(MQL5_OPTIMIZATION))
{
    MessageBox("Необходимо запускать в тестере стратегий в режиме оптимизации!");
//--- завершаем работу эксперта досрочно и снимаем с графика
    return(INIT_FAILED);
}
//--- успешное завершение инициализации
    return(INIT_SUCCEEDED);
}

//+-----+
//| TesterInit function
//+-----+
void OnTesterInit()
{
//--- пример для input-параметра типа long
string name="Input1";
bool enable;
long par1,par1_start,par1_step,par1_stop;
```

```

ParameterGetRange(name,enable,par1,par1_start,par1_step,par1_stop);
Print("Первый параметр");
PrintFormat("%s=%d enable=%s from %d to %d with step=%d",
           name,par1,(string)enable,par1_start,par1_stop,par1_step);
//--- пример для input-параметра типа double
name="Input2";
double par2,par2_start,par2_step,par2_stop;
ParameterGetRange(name,enable,par2,par2_start,par2_step,par2_stop);
Print("Второй параметр");
PrintFormat("%s=%G enable=%s from %G to %G with step=%G",
           name,par2,(string)enable,par2_start,par2_stop,par2_step);

//--- пример для input-параметра типа bool
name="Input3";
long par3,par3_start,par3_step,par3_stop;
ParameterGetRange(name,enable,par3,par3_start,par3_step,par3_stop);
Print("Третий параметр");
PrintFormat("%s=%s enable=%s from %s to %s",
           name,(string)par3,(string)enable,
           (string)par3_start,(string)par3_stop);
//--- пример для input-параметра типа перечисление
name="Input4";
long par4,par4_start,par4_step,par4_stop;
ParameterGetRange(name,enable,par4,par4_start,par4_step,par4_stop);
Print("Четвертый параметр");
PrintFormat("%s=%s enable=%s from %s to %s",
           name,EnumToString((ENUM_DAY_OF_WEEK)par4),(string)enable,
           EnumToString((ENUM_DAY_OF_WEEK)par4_start),
           EnumToString((ENUM_DAY_OF_WEEK)par4_stop));
}

//-----+
//| TesterDeinit function
//-----+
void OnTesterDeinit()
{
//--- это сообщение выйдет по окончании оптимизации
Print(__FUNCTION__," Optimization completed");
}

```

## ParameterSetRange

Устанавливает правила использования [input-переменной](#) при оптимизации эксперта в тестере стратегий: значение, шаг изменения, начальное и конечное значения. Существует 2 варианта функции.

### 1. Установка значений для input-параметра целого типа

```
bool ParameterSetRange(
    const string name,           // имя параметра (input-переменной)
    bool enable,                 // разрешить оптимизацию параметра
    long value,                  // значение параметра
    long start,                 // начальное значение
    long step,                  // шаг изменения
    long stop                   // конечное значение
);
```

### 2. Установка значений для input-параметра вещественного типа

```
bool ParameterSetRange(
    const string name,           // имя параметра (input-переменной)
    bool enable,                 // разрешить оптимизацию параметра
    double value,                // значение параметра
    double start,                // начальное значение
    double step,                 // шаг изменения
    double stop                  // конечное значение
);
```

## Параметры

*name*

[in] Идентификатор переменной [input](#) или [sinput](#). Такие переменные являются внешними параметрами программы, значения которых можно задавать при запуске.

*enable*

[in] Разрешить данный параметр для перебора значений в процессе оптимизации в тестере стратегий.

*value*

[in] Значение параметра.

*start*

[in] Начальное значение параметра при оптимизации.

*step*

[in] Шаг изменения параметра при переборе его значений.

*stop*

[in] Конечное значение параметра при оптимизации.

## Возвращаемое значение

Возвращает true в случае успешного выполнения, иначе false. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

#### Примечание

Функция может вызываться только из обработчика [OnTesterInit\(\)](#) при запуске оптимизации в тестере стратегий. Предназначена для установки диапазона и шага изменения параметра, в том числе позволяет полностью исключать этот параметр из процедуры оптимизации, несмотря на настройки в тестере стратегий. Также позволяет использовать в оптимизации даже переменные, объявленные с модификатором sinput.

Функция ParameterSetRange() позволяет создавать собственные сценарии оптимизации эксперта в тестере стратегий в зависимости от значений его ключевых параметров, то есть включать либо исключать из оптимизации требуемые входные параметры, а также задавать требуемые диапазон и шаг изменения.

## Работа с событиями

Функции для работы с пользовательскими событиями и событиями таймера. Кроме этих функций существуют также специальные функции, для обработки [предопределенных событий](#).

Функция	Действие
<a href="#">EventSetMillisecondTimer</a>	Запускает генератор событий таймера высокого разрешения с периодом менее 1 секунды для текущего графика
<a href="#">EventSetTimer</a>	Запускает генератор событий таймера с указанной периодичностью для текущего графика
<a href="#">EventKillTimer</a>	Останавливает на текущем графике генерацию событий по таймеру
<a href="#">EventChartCustom</a>	Генерирует пользовательское событие для указанного графика

Смотри также

[Типы событий графика](#)

## EventSetMillisecondTimer

Указывает клиентскому терминалу, что для данного эксперта или индикатора необходимо генерировать события [таймера](#) с периодичностью менее одной секунды.

```
bool EventSetMillisecondTimer(
    int milliseconds      // количество миллисекунд
);
```

### Параметры

*milliseconds*

[in] Количество миллисекунд, определяющее периодичность возникновения событий от таймера.

### Возвращаемое значение

В случае успешного выполнения возвращает true, иначе - false. Для получения кода [ошибки](#) нужно вызвать функцию [GetLastError\(\)](#).

### Примечание

Эта функция предназначена для тех случаев, когда требуется таймер высокого разрешения, то есть нужно получать события таймера чаще, чем один раз в секунду. Если вам достаточно обычного таймера с периодом более 1 секунды, то используйте [EventSetTimer\(\)](#).

В тестере стратегий используется минимальный интервал в 1000 миллисекунд. В общем случае при уменьшении периода таймера увеличивается время тестирования, так как возрастает количество вызовов обработчика событий таймера. При работе в режиме реального времени события таймера генерируются не чаще 1 раза в 10-16 миллисекунд, что связано с аппаратными ограничениями.

Обычно эта функция должна вызываться из функции [OnInit\(\)](#) или в [конструкторе](#) класса. Для того чтобы обрабатывать события, приходящие от таймера, эксперт или индикатор должен иметь функцию [OnTimer\(\)](#).

Каждый эксперт и каждый индикатор работает со своим таймером и получает события только от него. При завершении работы mq5-программы таймер уничтожается принудительно, если он был создан, но не был отключен функцией [EventKillTimer\(\)](#).

Для каждой программы может быть запущено не более одного таймера. Каждая mq5-программа и каждый график имеют свою собственную очередь событий, куда складываются все вновь поступающие события. Если в очереди уже есть событие [Timer](#) либо это событие находится в состоянии обработки, то новое событие Timer в очередь mq5-программы не ставится.

## EventSetTimer

Указывает клиентскому терминалу, что для данного эксперта или индикатора необходимо генерировать события от [таймера](#) с указанной периодичностью.

```
bool EventSetTimer(  
    int seconds           // количество секунд  
);
```

### Параметры

*seconds*

[in] Количество секунд, определяющее периодичность возникновения событий от таймера.

### Возвращаемое значение

В случае успешного выполнения возвращает true, иначе false. Для получения кода [ошибки](#) нужно вызвать функцию [GetLastError\(\)](#).

### Примечание

Обычно, эта функция должна вызываться из функции [OnInit\(\)](#) или в [конструкторе](#) класса. Для того чтобы обрабатывать события, приходящие от таймера, эксперт или индикатор должен иметь функцию [OnTimer\(\)](#).

Каждый эксперт и каждый индикатор работает со своим таймером, и получает события только от него. При завершении работы mq5-программы таймер уничтожается принудительно, если он был создан, но не был отключен функцией [EventKillTimer\(\)](#).

Для каждой программы может быть запущено не более одного таймера. Каждая mq5-программа и каждый график имеют свою собственную очередь событий, куда складываются все вновь поступающие события. Если в очереди уже есть событие [Timer](#) либо это событие находится в состоянии обработки, то новое событие Timer в очередь mq5-программы не ставится.

## EventKillTimer

Указывает клиентскому терминалу, что необходимо остановить генерацию событий от [таймера](#) для данного эксперта или индикатора.

```
void EventKillTimer();
```

### Возвращаемое значение

Нет возвращаемого значения.

### Примечание

Обычно эта функция должна вызываться из функции [OnDeinit\(\)](#) в том случае, если в функции [OnInit\(\)](#) была вызвана функция [EventSetTimer\(\)](#). Либо должна вызываться из деструктора класса, если в [конструкторе](#) этого класса вызывается функция `EventSetTimer()`.

Каждый эксперт и каждый индикатор работает со своим таймером, и получает события только от него. При завершении работы MQL5-программы таймер уничтожается принудительно, если он был создан, но не был отключен функцией `EventKillTimer()`.

## EventChartCustom

Генерирует пользовательское событие для указанного графика.

```
bool EventChartCustom(
    long chart_id,           // идентификатор графика-получателя события
    ushort custom_event_id,  // идентификатор события
    long lparam,             // параметр типа long
    double dparam,            // параметр типа double
    string sparam             // строковый параметр события
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*custom\_event\_id*

[in] Идентификатор пользовательского события. Этот идентификатор автоматически добавляется к значению [CHARTEVENT\\_CUSTOM](#) и преобразуется к целому типу.

*lparam*

[in] Параметр события типа long, передаваемый функции [OnChartEvent](#).

*dparam*

[in] Параметр события типа double, передаваемый функции OnChartEvent.

*sparam*

[in] Параметр события типа string, передаваемый функции OnChartEvent. Если строка имеет длину больше, чем 63 символа, то строка усекается.

### Возвращаемое значение

Возвращает true в случае удачной постановки пользовательского события в очередь событий графика - получателя события. В случае ошибки возвращает false, для получения кода ошибки используйте [GetLastError\(\)](#).

### Примечание

Эксперт или индикатор, прикрепленный к указанному графику, обрабатывает данное событие при помощи функции [OnChartEvent](#)(int event\_id, long& lparam, double& dparam, string& sparam).

Для каждого типа события входные параметры функции [OnChartEvent\(\)](#) имеют определенные значения, которые необходимы для обработки этого события. В таблице перечислены события и значения, которые передаются через параметры.

Событие	Значение параметра id	Значение параметра lparam	Значение параметра dparam	Значение параметра sparam
Событие нажатия клавиатуры	CHARTEVENT_KEYDOWN	код нажатой клавиши	Количество нажатий клавиши, сгенер	Строковое значение битовой маски,

			ированных за время её удержания нажатом состояния	описывающее статус кнопок клавиатуры
События мыши (если для графика установлено свойство <b>CHART_EVENT_MOUSE_MOVE=true</b> )	CHARTEVENT_MOUSE_MOVE	X координата	Y координата	Строковое значение битовой маски, описывающее статус кнопок мыши
Событие создания графического объекта (если для графика установлено свойство <b>CHART_EVENT_OBJECT_CREATE=true</b> )	CHARTEVENT_OBJECT_CREATE	—	—	Имя созданного графического объекта
Событие изменения свойств объекта через диалог свойств	CHARTEVENT_OBJECT_CHANGE	—	—	Имя измененного графического объекта
Событие удаления графического объекта (если для графика установлено свойство <b>CHART_EVENT_OBJECT_DELETE=true</b> )	CHARTEVENT_OBJECT_DELETE	—	—	Имя удаленного графического объекта
Событие щелчка мыши на графике	CHARTEVENT_CLICK	X координата	Y координата	—
Событие щелчка мыши на графическом объекте	CHARTEVENT_OBJECT_CLICK	X координата	Y координата	Имя графического объекта, на котором произошло событие

Событие перемещения графического объекта при помощи мыши	CHARTEVENT_OBJECT_DRAG	—	—	Имя перемещенного графического объекта
Событие окончания редактирования текста в поле ввода графического объекта "Поле ввода"	CHARTEVENT_OBJECT_ENDEDIT	—	—	Имя графического объекта "Поле ввода", в котором завершилось редактирование текста
Событие изменения графика	CHARTEVENT_CHART_CHANGE	—	—	—
Пользовательское событие с номером N	CHARTEVENT_CUSTOM+N	Значение, заданное функцией EventChartCustom()	Значение, заданное функцией EventChartCustom()	Значение, заданное функцией EventChartCustom()

Пример:

```
//+-----+
//| ButtonClickExpert.mq5 |
//| Copyright 2009, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

string buttonID="Button";
string labelID="Info";
int broadcastEventID=5000;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- создадим кнопку, для передачи пользовательских событий
ObjectCreate(0,buttonID,OBJ_BUTTON,0,100,100);
ObjectSetInteger(0,buttonID,OBJPROP_COLOR,clrWhite);
ObjectSetInteger(0,buttonID,OBJPROP_BGCOLOR,clrGray);
ObjectSetInteger(0,buttonID,OBJPROP_XDISTANCE,100);
ObjectSetInteger(0,buttonID,OBJPROP_YDISTANCE,100);
ObjectSetInteger(0,buttonID,OBJPROP_XSIZE,200);
```

```

ObjectSetInteger(0,buttonID,OBJPROP_YSIZE,50);
ObjectSetString(0,buttonID,OBJPROP_FONT,"Arial");
ObjectSetString(0,buttonID,OBJPROP_TEXT,"Кнопка");
ObjectSetInteger(0,buttonID,OBJPROP_FONTSIZE,10);
ObjectSetInteger(0,buttonID,OBJPROP_SELECTABLE,0);

//--- создадим метку для вывода информации
ObjectCreate(0,labelID,OBJ_LABEL,0,100,100);
ObjectSetInteger(0,labelID,OBJPROP_COLOR,clrRed);
ObjectSetInteger(0,labelID,OBJPROP_XDISTANCE,100);
ObjectSetInteger(0,labelID,OBJPROP_YDISTANCE,50);
ObjectSetString(0,labelID,OBJPROP_FONT,"Trebuchet MS");
ObjectSetString(0,labelID,OBJPROP_TEXT,"Нет информации");
ObjectSetInteger(0,labelID,OBJPROP_FONTSIZE,20);
ObjectSetInteger(0,labelID,OBJPROP_SELECTABLE,0);

//---
return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function           |
//+-----+
void OnDeinit(const int reason)
{
//---
ObjectDelete(0,buttonID);
ObjectDelete(0,labelID);
}

//+-----+
//| Expert tick function                         |
//+-----+
void OnTick()
{
//---

}

//+-----+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
{
//--- проверим событие на нажатие кнопки мышки
if(id==CHARTEVENT_OBJECT_CLICK)
{
    string clickedChartObject=sparam;
//--- если нажатие на объекте с именем buttonID
if(clickedChartObject==buttonID)
{
    //--- состояние кнопки - нажата кнопка или нет
}
}
}

```

```

bool selected=ObjectGetInteger(0,buttonID,OBJPROP_STATE);
//--- выведем в лог отладочное сообщение
Print("Кнопка нажата = ",selected);
int customEventID; // номер пользовательского события для отправки
string message; // сообщение для отправки в событии
//--- если кнопка нажата
if(selected)
{
    message="Кнопка нажата";
    customEventID=CHARTEVENT_CUSTOM+1;
}
else // кнопка не нажата
{
    message="Кнопка отжата";
    customEventID=CHARTEVENT_CUSTOM+999;
}
//--- отправим пользовательское событие "своему" графику
EventChartCustom(0,customEventID-CHARTEVENT_CUSTOM,0,0,message);
//--- отправим сообщение всем открытым графикам
BroadcastEvent(ChartID(),0,"Broadcast Message");
//--- отладочное сообщение
Print("Отправлено событие с ID = ",customEventID);
}
ChartRedraw(); // принудительно перерисуем все объекты на графике
}

//--- проверим событие на принадлежность к пользовательским событиям
if(id>CHARTEVENT_CUSTOM)
{
    if(id==broadcastEventID)
    {
        Print("Получили широковещательное сообщение от графика с id = "+lparam);
    }
    else
    {
        //--- прочитаем текстовое сообщение в событии
        string info=sparam;
        Print("Обрабатывается ПОЛЬЗОВАТЕЛЬСКОЕ событие с ID = ",id);
        //--- выведем сообщение в метке
        ObjectSetString(0,labelID,OBJPROP_TEXT,sparam);
        ChartRedraw(); // принудительно перерисуем все объекты на графике
    }
}
}

//-----+
//| послать широковещательное сообщение всем открытым графикам |+
//-----+
void BroadcastEvent(long lparam,double dparam,string sparam)
{
}

```

```
int eventID=broadcastEventID-CHARTEVENT_CUSTOM;
long currChart=ChartFirst();
int i=0;
while(i<CHARTS_MAX)           // у нас наверняка не больше CHARTS_MAX открыты
{
    EventChartCustom(currChart,eventID,lparam,dparam,sparam);
    currChart=ChartNext(currChart); // на основании предыдущего получим новый график
    if(currChart== -1) break;     // достигли конца списка графиков
    i++;                         // не забудем увеличить счетчик
}
//+-----+
```

#### Смотри также

[События клиентского терминала](#), [Функции обработки событий](#)

## Работа с OpenCL

Программы на [OpenCL](#) предназначены для выполнения вычислений на видеокартах с поддержкой стандарта OpenCL 1.1 или выше. Современные видеокарты содержат сотни небольших специализированных процессоров, которые могут одновременно выполнять простые математические операции над входящими потоками данных. Язык OpenCL берёт на себя организацию таких параллельных вычислений и позволяет добиться огромного ускорения для некоторого класса задач.

На некоторых видеокартах по умолчанию отключен режим работы с числами типа [double](#), что приводит к возникновению ошибки компиляции 5105. Для включения режима поддержки чисел double в текст OpenCL-программы нужно добавить директиву [#pragma OPENCL EXTENSION cl\\_khr\\_fp64 : enable](#). Однако, если видеокарта не поддерживает double, то включение данной директивы не поможет.

Исходный код OpenCL рекомендуется писать в отдельных CL-файлах, которые затем можно подключать к MQL5-программе с помощью [ресурсных переменных](#).

Функции для выполнения программ на OpenCL:

Функция	Действие
<a href="#">CLHandleType</a>	Возвращает тип OpenCL хендла в виде значения из перечисления ENUM_OPENCL_HANDLE_TYPE
<a href="#">CLGetInfoInteger</a>	Возвращает значение целочисленного свойства для OpenCL-объекта или устройства
<a href="#">CLGetInfoString</a>	Возвращает строковое значение свойства для OpenCL-объекта или устройства
<a href="#">CLContextCreate</a>	Создает контекст OpenCL
<a href="#">CLContextFree</a>	Удаляет контекст OpenCL
<a href="#">CLGetDeviceInfo</a>	Получает свойство устройства из OpenCL драйвера
<a href="#">CLProgramCreate</a>	Создает OpenCL программу из исходного кода
<a href="#">CLProgramFree</a>	Удаляет OpenCL программу
<a href="#">CLKernelCreate</a>	Создает функцию запуска OpenCL
<a href="#">CLKernelFree</a>	Удаляет функцию запуска OpenCL
<a href="#">CLSetKernelArg</a>	Выставляет параметр для функции OpenCL
<a href="#">CLSetKernelArgMem</a>	Выставляет буфер OpenCL в качестве параметра функции OpenCL
<a href="#">CLSetKernelArgMemLocal</a>	Задаёт локальный буфер в качестве аргумента kernel-функции
<a href="#">CLBufferCreate</a>	Создает буфер OpenCL

<a href="#"><u>CLBufferFree</u></a>	Удаляет буфер OpenCL
<a href="#"><u>CLBufferWrite</u></a>	Записывает массив в буфер OpenCL и возвращает количество записанных элементов
<a href="#"><u>CLBufferRead</u></a>	Читает буфер OpenCL в массив и возвращает количество прочитанных элементов
<a href="#"><u>CLExecute</u></a>	Выполняет OpenCL программу
<a href="#"><u>CLExecutionStatus</u></a>	Возвращает состояние выполнения OpenCL программы

Смотри также

[OpenCL](#), [Ресурсы](#)

## CLHandleType

Возвращает тип OpenCL хендла в виде значения из перечисления ENUM\_OPENCL\_HANDLE\_TYPE.

```
ENUM_OPENCL_HANDLE_TYPE CLHandleType(
    int handle // хендл объекта OpenCL
);
```

### Параметры

*handle*

[in] Хендл на объект OpenCL: контекст, кернел, буфер или программа OpenCL.

### Возвращаемое значение

Тип хендла OpenCL в виде значения из перечисления [ENUM\\_OPENCL\\_HANDLE\\_TYPE](#).

### ENUM\_OPENCL\_HANDLE\_TYPE

Идентификатор	Описание
OPENCL_INVALID	Некорректный хендл
OPENCL_CONTEXT	Хендл контекста OpenCL
OPENCL_PROGRAM	Хендл программы OpenCL
OPENCL_KERNEL	Хендл кернела OpenCL
OPENCL_BUFFER	Хендл буфера OpenCL

## CLGetInfoInteger

Возвращает значение целочисленного свойства для OpenCL-объекта или устройства.

```
long CLGetInfoInteger(
    int handle, // хэндл объекта OpenCL или номер OpenCL устройства
    ENUM_OPENCL_PROPERTY_INTEGER prop // запрашиваемое свойство
);
```

### Параметры

*handle*

[in] Хэндл на объект OpenCL или номер OpenCL устройства. Нумерация OpenCL устройств начинается с нуля.

*prop*

[in] Тип запрашиваемого свойства из перечисления [ENUM\\_OPENCL\\_PROPERTY\\_INTEGER](#), значение которого нужно получить.

### Возвращаемое значение

Значение указанного свойства при успешном выполнении или -1 в случае ошибки. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

### ENUM\_OPENCL\_PROPERTY\_INTEGER

Идентификатор	Описание	Тип
CL_DEVICE_COUNT	Количество устройств с поддержкой OpenCL. Для этого свойства не требуется указание первого параметра, то есть можно передать нулевое значение для параметра <i>handle</i> .	int
CL_DEVICE_TYPE	Тип устройства	<a href="#">ENUM_CL_DEVICE_TYPE</a>
CL_DEVICE_VENDOR_ID	Уникальный идентификатор производителя	uint
CL_DEVICE_MAX_COMPUTE_UNITS	Количество параллельных вычисляемых задач в устройстве OpenCL. Одна рабочая группа выполняет одну вычислительную задачу. Минимальное значение равно 1	uint
CL_DEVICE_MAX_CLOCK_FREQUENCY	Максимальная установленная частота устройства в МГц.	uint
CL_DEVICE_GLOBAL_MEM_SIZE	Размер глобальной памяти устройства в байтах	ulong

CL_DEVICE_LOCAL_MEM_SIZE	Размер локальной памяти обрабатываемых данных (сцены) в байтах	uint
CL_BUFFER_SIZE	Реальный размер буфера OpenCL в байтах	ulong
CL_DEVICE_MAX_WORK_GROUP_SIZE	Общее количество локальных рабочих групп доступных для OpenCL устройства.	ulong
CL_KERNEL_WORK_GROUP_SIZE	Общее количество локальных рабочих групп, доступных для OpenCL программы.	ulong
CL_KERNEL_LOCAL_MEM_SIZE	Размер локальной памяти в байтах, которую использует OpenCL программа для всех параллельных задач в группе. Используйте CL_DEVICE_LOCAL_MEM_SIZE для получения доступного максимума	ulong
CL_KERNEL_PRIVATE_MEM_SIZE	Минимальный размер приватной памяти в байтах, используемый каждой задачей в кернеле OpenCL программы.	ulong

Перечисление ENUM\_CL\_DEVICE\_TYPE содержит возможные типы устройств с поддержкой OpenCL. Получить тип устройства можно по его номеру или хэнду объекта OpenCL с помощью вызова CLGetInfoInteger(handle\_or\_deviceN, CL\_DEVICE\_TYPE).

#### ENUM\_CL\_DEVICE\_TYPE

Идентификатор	Описание
CL_DEVICE_ACCELERATOR	Специализированный OpenCL ускоритель (например, IBM CELL Blade).
CL_DEVICE_CPU	Использование центрального процессора компьютера в качестве OpenCL устройства. Центральный процессор может иметь одно или более вычислительных ядер.
CL_DEVICE_GPU	OpenCL устройство на основе видеокарты.
CL_DEVICE_DEFAULT	OpenCL устройство по умолчанию. Устройством по умолчанию не может являться CL_DEVICE_TYPE_CUSTOM устройство.
CL_DEVICE_CUSTOM	Специализированные ускорители, которые не поддерживают программы на OpenCL C.

Пример:

```
void OnStart()
{
    int cl_ctx;
//--- инициализация OpenCL контекста
    if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
    {
        Print("OpenCL not found");
        return;
    }
//--- Выведем общую информацию об устройстве OpenCL
    Print("OpenCL type: ",EnumToString((ENUM_CL_DEVICE_TYPE)CLGetInfoInteger(cl_ctx,CL_DEVICE_TYPE)));
    Print("OpenCL vendor ID: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_VENDOR_ID));
    Print("OpenCL units: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_COMPUTE_UNITS));
    Print("OpenCL freq: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_CLOCK_FREQUENCY)," MHz");
    Print("OpenCL global mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_GLOBAL_MEM_SIZE)," byte");
    Print("OpenCL local mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_LOCAL_MEM_SIZE)," byte");
//---
}
```

## CLGetInfoString

Возвращает строковое значение свойства для OpenCL-объекта или устройства.

```
bool CLGetInfoString(
    int handle, // хэндл объекта OpenCL или номер OpenCL устройства
    ENUM_OPENCL_PROPERTY_STRING prop, // запрашиваемое свойство
    string& value // строка по ссылке
);
```

### Параметры

*handle*

[in] Хэндл на объект OpenCL или номер OpenCL устройства. Нумерация OpenCL устройств начинается с нуля.

*prop*

[in] Тип запрашиваемого свойства из перечисления [ENUM\\_OPENCL\\_PROPERTY\\_STRING](#), значение которого нужно получить.

*value*

[out] Стока для получения значения свойства.

### Возвращаемое значение

true при успешном выполнении или false в случае ошибки. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

### ENUM\_OPENCL\_PROPERTY\_STRING

Идентификатор	Описание
CL_PLATFORM_PROFILE	CL_PLATFORM_PROFILE - Профиль OpenCL. Имя профиля может быть одним из значений: <ul style="list-style-type: none"> <li>• FULL_PROFILE - реализация поддерживает OpenCL (функциональность определена как часть спецификации ядра и не требует дополнительных расширений для поддержки OpenCL);</li> <li>• EMBEDDED_PROFILE - реализация поддерживает OpenCL в виде дополнения. Дополненный профиль определяется как подмножество для каждой версии OpenCL.</li> </ul>
CL_PLATFORM_VERSION	Версия OpenCL
CL_PLATFORM_VENDOR	Название производителя устройства
CL_PLATFORM_EXTENSIONS	Список расширений, поддерживаемых платформой. Названия расширений должны поддерживаться всеми устройствами, связанными с данной платформой

CL_DEVICE_NAME	Название устройства
CL_DEVICE_VENDOR	Название производителя
CL_DRIVER_VERSION	Версия драйвера OpenCL в формате major_number.minor_number
CL_DEVICE_PROFILE	<p>Профиль OpenCL устройства. Имя профиля может быть одним из следующих значений:</p> <ul style="list-style-type: none"> <li>• FULL_PROFILE - реализация поддерживает OpenCL (функциональность определена как часть спецификации ядра и не требует дополнительных расширений для поддержки OpenCL);</li> <li>• EMBEDDED_PROFILE - реализация поддерживает OpenCL в виде дополнения. Дополненный профиль определяется как подмножество для каждой версии OpenCL.</li> </ul>
CL_DEVICE_VERSION	Версия OpenCL в формате "OpenCL<space><major_version.minor_version><space><vendor-specific information>"
CL_DEVICE_EXTENSIONS	<p>Список расширений, поддерживаемых устройством. Список может включать расширения, поддерживаемые производителем и содержать один или более одобренных имен:</p> <pre>cl_khr_int64_base_atomics cl_khr_int64_extended_atomics cl_khr_fp16 cl_khr_gl_sharing cl_khr_gl_event cl_khr_d3d10_sharing cl_khr_dx9_media_sharing cl_khr_d3d11_sharing</pre>
CL_DEVICE_BUILT_IN KERNELS	Список кернелов, поддерживаемых устройством, разделенных ";" .
CL_DEVICE_OPENCL_C VERSION	<p>Максимальная версия, поддерживаемая компилятором для данного устройства. Формат версии:</p> <pre>"OpenCL&lt;space&gt;C&lt;space&gt;&lt;major_version.minor_version&gt;&lt;space&gt;&lt;vendor-specific information&gt;"</pre>

Пример:

```
void OnStart()
{
    int cl_ctx;
    string str;
    //--- инициализация OpenCL контекста
```

```

if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
{
    Print("OpenCL not found");
    return;
}

//--- Выведем информацию о платформе
if(CLGetInfoString(cl_ctx,CL_PLATFORM_NAME,str))
    Print("OpenCL platform name: ",str);
if(CLGetInfoString(cl_ctx,CL_PLATFORM_VENDOR,str))
    Print("OpenCL platform vendor: ",str);
if(CLGetInfoString(cl_ctx,CL_PLATFORM_VERSION,str))
    Print("OpenCL platform ver: ",str);
if(CLGetInfoString(cl_ctx,CL_PLATFORM_PROFILE,str))
    Print("OpenCL platform profile: ",str);
if(CLGetInfoString(cl_ctx,CL_PLATFORM_EXTENSIONS,str))
    Print("OpenCL platform ext: ",str);

//--- Выведем информацию о самом устройстве
if(CLGetInfoString(cl_ctx,CL_DEVICE_NAME,str))
    Print("OpenCL device name: ",str);
if(CLGetInfoString(cl_ctx,CL_DEVICE_PROFILE,str))
    Print("OpenCL device profile: ",str);
if(CLGetInfoString(cl_ctx,CL_DEVICE_BUILT_IN_KERNELS,str))
    Print("OpenCL device kernels: ",str);
if(CLGetInfoString(cl_ctx,CL_DEVICE_EXTENSIONS,str))
    Print("OpenCL device ext: ",str);
if(CLGetInfoString(cl_ctx,CL_DEVICE_VENDOR,str))
    Print("OpenCL device vendor: ",str);
if(CLGetInfoString(cl_ctx,CL_DEVICE_VERSION,str))
    Print("OpenCL device ver: ",str);
if(CLGetInfoString(cl_ctx,CL_DEVICE_OPENCL_C_VERSION,str))
    Print("OpenCL open c ver: ",str);

//--- Выведем общую информацию об устройстве OpenCL
Print("OpenCL type: ",EnumToString((ENUM_CL_DEVICE_TYPE)CLGetInfoInteger(cl_ctx,CL_DEVICE_TYPE)));
Print("OpenCL vendor ID: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_VENDOR_ID));
Print("OpenCL units: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_COMPUTE_UNITS));
Print("OpenCL freq: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_CLOCK_FREQUENCY));
Print("OpenCL global mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_GLOBAL_MEM_SIZE));
Print("OpenCL local mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_LOCAL_MEM_SIZE));

//---
}

```

## CLContextCreate

Создает контекст OpenCL и возвращает хендл на него.

```
int CLContextCreate(
    int device // порядковый номер OpenCL устройства или макрос
);
```

### Параметры

*device*

[in] Номер OpenCL-устройства в системе по порядку. Вместо конкретного номера можно указать одно из значений:

- CL\_USE\_ANY - разрешается использовать любое доступное устройство с поддержкой OpenCL;
- CL\_USE\_CPU\_ONLY - разрешается использовать только эмуляцию OpenCL на CPU;
- CL\_USE\_GPU\_ONLY - эмуляция OpenCL запрещена и разрешается использовать только специализированные устройства с поддержкой OpenCL (видеокарты).

### Возвращаемое значение

Хендл на контекст OpenCL при успешном создании, или -1 в случае ошибки. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

## CLContextFree

Удаляет контекст OpenCL.

```
void CLContextFree(
    int context // хэндл на контекст OpenCL
);
```

### Параметры

*context*

[in] Хэндл контекста OpenCL.

### Возвращаемое значение

Нет. В случае внутренней ошибки изменяется значение [\\_LastError](#). Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

## CLGetDeviceInfo

Получает свойство устройства из OpenCL драйвера.

```
bool CLGetDeviceInfo(
    int     handle,           // хендл устройства OpenCL
    int     property_id,      // идентификатор запрашиваемого свойства
    uchar& data[],          // массив для получения данных
    uint&   size             // смещение в массиве в элементах, по умолчанию 0
);
```

### Параметры

*handle*

[in] Номер OpenCL устройства или хендл OpenCL, созданный функцией [CLContextCreate\(\)](#).

*property\_id*

[in] Идентификатор свойства, которое необходимо получить об устройстве OpenCL. Может быть одним из предопределенных значений, перечисленных в [таблице ниже](#).

*data[]*

[out] Массив для получения данных о запрашиваемом свойстве.

*size*

[out] Размер полученных данных в массиве *data[]*.

### Возвращаемое значение

true при успешном выполнении или false в случае ошибки. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

### Примечание

Для одномерных массивов номер элемента, с которого начинается чтение данных для записи в буфер OpenCL, вычисляется с учётом флага [AS\\_SERIES](#).

### Перечень допустимых идентификаторов свойств OpenCL устройства

Точное описание свойства и его назначение можно найти на [официальном сайте OpenCL](#).

Идентификатор	Значение
CL_DEVICE_TYPE	0x1000
CL_DEVICE_VENDOR_ID	0x1001
CL_DEVICE_MAX_COMPUTE_UNITS	0x1002
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS	0x1003
CL_DEVICE_MAX_WORK_GROUP_SIZE	0x1004
CL_DEVICE_MAX_WORK_ITEM_SIZES	0x1005
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR	0x1006

CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT	0x1007
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT	0x1008
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG	0x1009
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT	0x100A
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE	0x100B
CL_DEVICE_MAX_CLOCK_FREQUENCY	0x100C
CL_DEVICE_ADDRESS_BITS	0x100D
CL_DEVICE_MAX_READ_IMAGE_ARGS	0x100E
CL_DEVICE_MAX_WRITE_IMAGE_ARGS	0x100F
CL_DEVICE_MAX_MEM_ALLOC_SIZE	0x1010
CL_DEVICE_IMAGE2D_MAX_WIDTH	0x1011
CL_DEVICE_IMAGE2D_MAX_HEIGHT	0x1012
CL_DEVICE_IMAGE3D_MAX_WIDTH	0x1013
CL_DEVICE_IMAGE3D_MAX_HEIGHT	0x1014
CL_DEVICE_IMAGE3D_MAX_DEPTH	0x1015
CL_DEVICE_IMAGE_SUPPORT	0x1016
CL_DEVICE_MAX_PARAMETER_SIZE	0x1017
CL_DEVICE_MAX_SAMPLERS	0x1018
CL_DEVICE_MEM_BASE_ADDR_ALIGN	0x1019
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE	0x101A
CL_DEVICE_SINGLE_FP_CONFIG	0x101B
CL_DEVICE_GLOBAL_MEM_CACHE_TYPE	0x101C
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE	0x101D
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE	0x101E
CL_DEVICE_GLOBAL_MEM_SIZE	0x101F
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE	0x1020
CL_DEVICE_MAX_CONSTANT_ARGS	0x1021
CL_DEVICE_LOCAL_MEM_TYPE	0x1022
CL_DEVICE_LOCAL_MEM_SIZE	0x1023

CL_DEVICE_ERROR_CORRECTION_SUPPORT	0x1024
CL_DEVICE_PROFILING_TIMER_RESOLUTION	0x1025
CL_DEVICE_ENDIAN_LITTLE	0x1026
CL_DEVICE_AVAILABLE	0x1027
CL_DEVICE_COMPILER_AVAILABLE	0x1028
CL_DEVICE_EXECUTION_CAPABILITIES	0x1029
CL_DEVICE_QUEUE_PROPERTIES	0x102A
CL_DEVICE_NAME	0x102B
CL_DEVICE_VENDOR	0x102C
CL_DRIVER_VERSION	0x102D
CL_DEVICE_PROFILE	0x102E
CL_DEVICE_VERSION	0x102F
CL_DEVICE_EXTENSIONS	0x1030
CL_DEVICE_PLATFORM	0x1031
CL_DEVICE_DOUBLE_FP_CONFIG	0x1032
CL_DEVICE_PREFERRED_VECTOR_WIDTH_HALF	0x1034
CL_DEVICE_HOST_UNIFIED_MEMORY	0x1035
CL_DEVICE_NATIVE_VECTOR_WIDTH_CHAR	0x1036
CL_DEVICE_NATIVE_VECTOR_WIDTH_SHORT	0x1037
CL_DEVICE_NATIVE_VECTOR_WIDTH_INT	0x1038
CL_DEVICE_NATIVE_VECTOR_WIDTH_LONG	0x1039
CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT	0x103A
CL_DEVICE_NATIVE_VECTOR_WIDTH_DOUBLE	0x103B
CL_DEVICE_NATIVE_VECTOR_WIDTH_HALF	0x103C
CL_DEVICE_OPENCL_C_VERSION	0x103D
CL_DEVICE_LINKER_AVAILABLE	0x103E
CL_DEVICE_BUILT_IN KERNELS	0x103F
CL_DEVICE_IMAGE_MAX_BUFFER_SIZE	0x1040
CL_DEVICE_IMAGE_MAX_ARRAY_SIZE	0x1041
CL_DEVICE_PARENT_DEVICE	0x1042
CL_DEVICE_PARTITION_MAX_SUB_DEVICES	0x1043

CL_DEVICE_PARTITION_PROPERTIES	0x1044
CL_DEVICE_PARTITION_AFFINITY_DOMAIN	0x1045
CL_DEVICE_PARTITION_TYPE	0x1046
CL_DEVICE_REFERENCE_COUNT	0x1047
CL_DEVICE_PREFERRED_INTEROP_USER_SYNC	0x1048
CL_DEVICE_PRINTF_BUFFER_SIZE	0x1049
CL_DEVICE_IMAGE_PITCH_ALIGNMENT	0x104A
CL_DEVICE_IMAGE_BASE_ADDRESS_ALIGNMENT	0x104B

Пример:

```
void OnStart()
{
//---
    int dCount= CLGetInfoInteger(0,CL_DEVICE_COUNT);
    for(int i = 0; i<dCount; i++)
    {
        int clCtx=CLContextCreate(i);
        if(clCtx == -1)
            Print("ERROR in CLContextCreate");
        string device;
        CLGetInfoString(clCtx,CL_DEVICE_NAME,device);
        Print(i+": ",device);
        uchar data[1024];
        uint size;
        CLGetDeviceInfo(clCtx,CL_DEVICE_VENDOR,data,size);
        Print("size = ",size);
        string str=CharArrayToString(data);
        Print(str);
    }
//--- пример вывода в журнал Эксперты
// 2013.07.24 10:50:48      opencl (EURUSD,H1)      2: Advanced Micro Devices, Inc.
// 2013.07.24 10:50:48      opencl (EURUSD,H1)      size = 32
// 2013.07.24 10:50:48      opencl (EURUSD,H1)      Tahiti
// 2013.07.24 10:50:48      opencl (EURUSD,H1)      Intel(R) Corporation
// 2013.07.24 10:50:48      opencl (EURUSD,H1)      size = 21
// 2013.07.24 10:50:48      opencl (EURUSD,H1)      1:           Intel(R) Core(TM) i7-3770K
// 2013.07.24 10:50:48      opencl (EURUSD,H1)      NVIDIA Corporation
// 2013.07.24 10:50:48      opencl (EURUSD,H1)      size = 19
// 2013.07.24 10:50:48      opencl (EURUSD,H1)      0: GeForce GTX 580
}
```

## CLProgramCreate

Создает OpenCL программу из исходного кода.

```
int CLProgramCreate(
    int         context,      // хендл на контекст OpenCL
    const string source       // исходный код
);
```

Перегруженная версия функции создает OpenCL программу и записывает сообщения компилятора в переданную строку.

```
int CLProgramCreate(
    int         context,      // хендл на контекст OpenCL
    const string source,     // исходный код
    string     &build_log    // строка для получения лога компиляции
);
```

### Параметры

*context*

[in] Хендл контекста OpenCL.

*source*

[in] Стока с исходным кодом OpenCL программы.

*&build\_log*

[in] Стока для получения сообщений компилятора OpenCL.

### Возвращаемое значение

Хендл на объект OpenCL при успешном выполнении. В случае ошибки возвращает -1. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#) и выводите значения строки *build\_log*.

### Примечание

На данный момент предусмотрены следующие коды ошибок:

- **ERR\_OPENCL\_INVALID\_HANDLE** - невалидный хендл на context OpenCL,
- **ERR\_INVALID\_PARAMETER** - невалидный строковой параметр,
- **ERR\_NOT\_ENOUGH\_MEMORY** - недостаточно памяти для завершения операции,
- **ERR\_OPENCL\_PROGRAM\_CREATE** - внутренняя ошибка OpenCL или ошибка компиляции.

На некоторых видеокартах по умолчанию отключен режим работы с числами типа [double](#), что приводит к возникновению ошибки компиляции 5105. Для включения режима поддержки чисел [double](#) в текст OpenCL-программы нужно добавить директиву [#pragma OPENCL EXTENSION cl\\_khr\\_fp64 : enable](#). Однако, если видеокарта не поддерживает [double](#), то включение данной директивы не поможет.

### Пример:

```
//+-----+
//| OpenCL kernel |
```

```

//+-----+
const string
cl_src=
    //--- by default some GPU doesn't support doubles
    //--- cl_khr_fp64 directive is used to enable work with doubles
    "#pragma OPENCL EXTENSION cl_khr_fp64 : enable      \r\n"
    //--- OpenCL kernel function
    "__kernel void Test_GPU(__global double *data,      \r\n"
    "                      const int N,                  \r\n"
    "                      const int total_arrays) \r\n"
    "  {                                \r\n"
    "    uint kernel_index=get_global_id(0);          \r\n"
    "    if (kernel_index>total_arrays) return;        \r\n"
    "    uint local_start_offset=kernel_index*N;       \r\n"
    "    for(int i=0; i<N; i++)                      \r\n"
    "    {                                \r\n"
    "      data[i+local_start_offset] *= 2.0;          \r\n"
    "    }                                \r\n"
    "  }                                \r\n"
    "\r\n";
//+-----+
//| Test_CPU
//+-----+
bool Test_CPU(double &data[],const int N,const int id,const int total_arrays)
{
//--- check array size
    if(ArraySize(data)==0) return(false);
//--- check array index
    if(id>total_arrays) return(false);
//--- calculate local offset for array with index id
    int local_start_offset=id*N;
//--- multiply elements by 2
    for(int i=0; i<N; i++)
    {
        data[i+local_start_offset]*=2.0;
    }
    return true;
}
//---
#define ARRAY_SIZE 100 // size of the array
#define TOTAL_ARRAYS 5 // total arrays
//--- OpenCL handles
int cl_ctx; // OpenCL context handle
int cl_prg; // OpenCL program handle
int cl_krn; // OpenCL kernel handle
int cl_mem; // OpenCL buffer handle
//---
double DataArray1[]; // data array for CPU calculation
double DataArray2[]; // data array for GPU calculation
//+-----+

```

```

//| Script program start function
//+-----+
int OnStart()
{
    //--- initialize OpenCL objects
    //--- create OpenCL context
    if((cl_ctx=CLContextCreate())==INVALID_HANDLE)
    {
        Print("OpenCL not found. Error=",GetLastError());
        return(1);
    }
    //--- create OpenCL program
    if((cl_prg=CLProgramCreate(cl_ctx,cl_src))==INVALID_HANDLE)
    {
        CLContextFree(cl_ctx);
        Print("OpenCL program create failed. Error=",GetLastError());
        return(1);
    }
    //--- create OpenCL kernel
    if((cl_krn=CLKernelCreate(cl_prg,"Test_GPU"))==INVALID_HANDLE)
    {
        CLProgramFree(cl_prg);
        CLContextFree(cl_ctx);
        Print("OpenCL kernel create failed. Error=",GetLastError());
        return(1);
    }
    //--- create OpenCL buffer
    if((cl_mem=CLBufferCreate(cl_ctx,ARRAY_SIZE*TOTAL_ARRAYS*sizeof(double),CL_MEM_READ_WRITE))!=cl_mem)
    {
        CLKernelFree(cl_krn);
        CLProgramFree(cl_prg);
        CLContextFree(cl_ctx);
        Print("OpenCL buffer create failed. Error=",GetLastError());
        return(1);
    }
    //--- set OpenCL kernel constant parameters
    CLSetKernelArgMem(cl_krn,0,cl_mem);
    CLSetKernelArg(cl_krn,1,ARRAY_SIZE);
    CLSetKernelArg(cl_krn,2,TOTAL_ARRAYS);
    //--- prepare data arrays
    ArrayResize(DataArray1,ARRAY_SIZE*TOTAL_ARRAYS);
    ArrayResize(DataArray2,ARRAY_SIZE*TOTAL_ARRAYS);
    //--- fill arrays with data
    for(int j=0; j<TOTAL_ARRAYS; j++)
    {
        //--- calculate local start offset for jth array
        uint local_offset=j*ARRAY_SIZE;
        //--- prepare array with index j
        for(int i=0; i<ARRAY_SIZE; i++)

```

```

    {
        //--- fill arrays with function MathCos(i+j);
        DataArray1[i+local_offset]=MathCos(i+j);
        DataArray2[i+local_offset]=MathCos(i+j);
    }
};

//--- test CPU calculation
for(int j=0; j<TOTAL_ARRAYS; j++)
{
    //--- calculation of the array with index j
    Test_CPU(DataArray1,ARRAY_SIZE,j,TOTAL_ARRAYS);
}

//--- prepare CLExecute params
uint offset[]={0};
//--- global work size
uint work[]={TOTAL_ARRAYS};
//--- write data to OpenCL buffer
CLBufferWrite(cl_mem,DataArray2);
//--- execute OpenCL kernel
CLEexecute(cl_krn,1,offset,work);
//--- read data from OpenCL buffer
CLBufferRead(cl_mem,DataArray2);
//--- total error
double total_error=0;
//--- compare results and calculate error
for(int j=0; j<TOTAL_ARRAYS; j++)
{
    //--- calculate local offset for jth array
    uint local_offset=j*ARRAY_SIZE;
    //--- compare the results
    for(int i=0; i<ARRAY_SIZE; i++)
    {
        double v1=DataArray1[i+local_offset];
        double v2=DataArray2[i+local_offset];
        double delta=MathAbs(v2-v1);
        total_error+=delta;
        //--- show first and last arrays
        if((j==0) || (j==TOTAL_ARRAYS-1))
            PrintFormat("array %d of %d, element [%d]: %f, %f, [error]=%f",j+1,TOTAL_
    }
}
PrintFormat("Total error: %f",total_error);
//--- delete OpenCL objects
//--- free OpenCL buffer
CLBufferFree(cl_mem);
//--- free OpenCL kernel
CLKernelFree(cl_krn);
//--- free OpenCL program
CLProgramFree(cl_prg);

```

```
//--- free OpenCL context
CLContextFree(cl_ctx);
//---
return(0);
}
```

## CLProgramFree

Удаляет OpenCL программу.

```
void CLProgramFree(
    int program      // хэндл на объект OpenCL
);
```

### Параметры

*program*

[in] Хэндл объекта OpenCL.

### Возвращаемое значение

Нет. В случае внутренней ошибки изменяется значение [\\_LastError](#). Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

## CLKernelCreate

Создает точку входа в программу OpenCL и возвращает хендл на неё.

```
int CLKernelCreate(
    int         program,           // хендл на объект OpenCL
    const string kernel_name      // имя кернела
);
```

### Параметры

*program*

[in] Хендл на объект программы OpenCL.

*kernel\_name*

[in] Имя функции кернел, то есть имя точки входа в соответствующей OpenCL-программе, с которой начинается выполнение.

### Возвращаемое значение

Хендл на объект OpenCL при успешном выполнении. В случае ошибки возвращает -1. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

### Примечание

На данный момент предусмотрены следующие коды ошибок:

- **ERR\_OPENCL\_INVALID\_HANDLE** - невалидный хендл OpenCL-программы,
- **ERR\_INVALID\_PARAMETER** - невалидный строковой параметр,
- **ERR\_OPENCL\_TOO\_LONG\_KERNEL\_NAME** - имя кернела содержит более 127 символов,
- **ERR\_OPENCL\_KERNEL\_CREATE** - внутренняя ошибка при создании объекта OpenCL.

## CLKernelFree

Удаляет функцию запуска OpenCL.

```
void CLKernelFree(
    int kernel // хендл на кернел OpenCL программы
);
```

### Параметры

*kernel\_name*

[in] Хендл объекта кернел.

### Возвращаемое значение

Нет. В случае внутренней ошибки изменяется значение [\\_LastError](#). Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

## CLSetKernelArg

Выставляет параметр для функции OpenCL.

```
bool CLSetKernelArg(  
    int   kernel,           // хендл на кернел OpenCL программы  
    uint  arg_index,        // номер аргумента OpenCL функции  
    void  arg_value         // исходный код  
) ;
```

### Параметры

*kernel*

[in] Хендл на кернел программы OpenCL.

*arg\_index*

[in] Номер аргумента функции, нумерация начинается с нуля.

*arg\_value*

[in] Значение аргумента функции.

### Возвращаемое значение

Возвращает true при успешном выполнении, в противном случае false. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

### Примечание

На данный момент предусмотрены следующие коды ошибок:

- ERR\_INVALID\_PARAMETER,
- ERR\_OPENCL\_INVALID\_HANDLE - невалидный хендл на кернел OpenCL,
- ERR\_OPENCL\_SET\_KERNEL\_PARAMETER - внутренняя ошибка OpenCL.

## CLSetKernelArgMem

Выставляет буфер OpenCL в качестве параметра функции OpenCL.

```
bool CLSetKernelArgMem(  
    int   kernel,           // хендл на кернел OpenCL программы  
    uint  arg_index,        // номер аргумента OpenCL функции  
    int   cl_mem_handle     // хендл буфера OpenCL  
);
```

### Параметры

*kernel*

[in] Хендл на кернел программы OpenCL.

*arg\_index*

[in] Номер аргумента функции, нумерация начинается с нуля.

*cl\_mem\_handle*

[in] Хендл на буфер OpenCL.

### Возвращаемое значение

Возвращает true при успешном выполнении, в противном случае false. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

## CLSetKernelArgMemLocal

Задаёт локальный буфер в качестве аргумента kernel-функции.

```
bool CLSetKernelArgMemLocal (
    int    kernel,           // хэндл на кернел OpenCL программы
    uint   arg_index,        // номер аргумента OpenCL функции
    ulong  local_mem_size   // размер буфера
);
```

### Параметры

*kernel*

[in] Хэндл на кернел программы OpenCL.

*arg\_index*

[in] Номер аргумента функции, нумерация начинается с нуля.

*local\_mem\_size*

[in] Размер буфера в байтах.

### Возвращаемое значение

Возвращает `true` при успешном выполнении, в противном случае `false`. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

## CLBufferCreate

Создает буфер OpenCL и возвращает хендл на него.

```
int CLBufferCreate(
    int   context,      // хендл на контекст OpenCL
    uint  size,         // размер буфера
    uint  flags         // комбинация флагов, задающие свойства буфера
);
```

### Параметры

*context*

[in] Хендл на context OpenCL.

*size*

[in] Размер буфера в байтах.

*flags*

[in] Свойства буфера, задаваемые через комбинацию флагов: CL\_MEM\_READ\_WRITE, CL\_MEM\_WRITE\_ONLY, CL\_MEM\_READ\_ONLY, CL\_MEM\_ALLOC\_HOST\_PTR.

### Возвращаемое значение

Хендл на буфер OpenCL при успешном выполнении. В случае ошибки возвращает -1. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

### Примечание

На данный момент предусмотрены следующие коды ошибок:

- ERR\_OPENCL\_INVALID\_HANDLE - невалидный хендл на контекст OpenCL,
- ERR\_NOT\_ENOUGH\_MEMORY - недостаточно памяти,
- ERR\_OPENCL\_BUFFER\_CREATE - внутренняя ошибка создания буфера.

## CLBufferFree

Удаляет буфер OpenCL.

```
void CLBufferFree(
    int buffer // хэндл на буфер OpenCL
);
```

### Параметры

*buffer*

[in] Хэндл на буфер OpenCL.

### Возвращаемое значение

Нет. В случае внутренней ошибки изменяется значение [\\_LastError](#). Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

## CLBufferWrite

Записывает массив в буфер OpenCL и возвращает количество записанных элементов.

```
uint CLBufferWrite(
    int      buffer,           // хендл на буфер OpenCL
    const void& data[],        // массив значений
    uint     buffer_offset=0,   // смещение в OpenCL буфере в байтах, по умолчанию 0
    uint     data_offset=0,     // смещение в массиве в элементах, по умолчанию 0
    uint     data_count=WHOLE_ARRAY // количество значений из массива для записи
);
```

### Параметры

*buffer*

[in] Хендл буфера OpenCL.

*data[]*

[in] Массив значений, которые необходимо записать в буфер OpenCL. Передается по ссылке.

*buffer\_offset*

[in] Смещение в OpenCL буфере в байтах, с которого начинается запись. По умолчанию запись идет с самого начала буфера.

*data\_offset*

[in] Индекс первого элемента массива, начиная с которого берутся значения из массива для записи в OpenCL буфер. По умолчанию значения берутся с самого начала массива.

*data\_count*

[in] Количество значений, которые нужно записать. По умолчанию все значения массива.

### Возвращаемое значение

Количество записанных элементов, в случае ошибки возвращается 0. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

### Примечание

Для одномерных массивов номер элемента, с которого начинается чтение данных для записи в буфер OpenCL, вычисляется с учётом флага [AS\\_SERIES](#).

Массив с размерностью два и более представляется как одномерный. В этом случае *data\_offset* - это количество элементов, которое следует пропустить в представлении, а не количество элементов в первой размерности.

## CLBufferRead

Читает буфер OpenCL в массив и возвращает количество прочитанных элементов.

```
uint CLBufferRead(
    int         buffer,           // хендл на буфер OpenCL
    const void&  data[],          // массив значений
    uint        buffer_offset=0,   // смещение в OpenCL буфере в байтах, по умолчанию 0
    uint        data_offset=0,     // смещение в массиве в элементах, по умолчанию 0
    uint        data_count=WHOLE_ARRAY // количество значений из буфера для чтения
);
```

### Параметры

*buffer*

[in] Хендл буфера OpenCL.

*data[]*

[in] Массив для получения значений из буфера OpenCL. Передается по ссылке.

*buffer\_offset*

[in] Смещение в OpenCL буфере в байтах, с которого начинается чтение. По умолчанию чтение начинается с начала буфера.

*data\_offset*

[in] Индекс первого элемента массива для записи значений буфера OpenCL. По умолчанию запись прочитанных значений в массив начинается с нулевого индекса.

*data\_count*

[in] Количество значений, которые нужно прочитать. По умолчанию читается весь буфер OpenCL.

### Возвращаемое значение

Количество прочитанных элементов, в случае ошибки возвращается 0. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

### Примечание

Для одномерных массивов номер элемента, в который начинается запись данных из буфера OpenCL, вычисляется с учётом флага [AS\\_SERIES](#).

Массив с размерностью два и более представляется как одномерный. В этом случае *data\_offset* - это количество элементов, которое следует пропустить в представлении, а не количество элементов в первой размерности.

## CLExecute

Выполняет OpenCL программу. Существует 3 варианта функции:

1. Запуск функции kernel на одном ядре

```
bool CLExecute(
    int      kernel           // хендл на кернел OpenCL программы
);
```

2. Запуск нескольких копий kernel (OpenCL функция) с описанием пространства задач

```
bool CLExecute(
    int      kernel,           // хендл на кернел OpenCL программы
    uint     work_dim,         // размерность пространства задач
    const uint& global_work_offset[], // начальное смещение в пространстве задач
    const uint& global_work_size[]   // общее количество задач
);
```

3. Запуск нескольких копий kernel (OpenCL функция) с описанием пространства задач и указанием размера локального подмножества задач в группе

```
bool CLExecute(
    int      kernel,           // хендл на кернел OpenCL программы
    uint     work_dim,          // размерность пространства задач
    const uint& global_work_offset[], // начальное смещение в пространстве задач
    const uint& global_work_size[], // общее количество задач
    const uint& local_work_size[]  // количество задач в локальной группе
);
```

### Параметры

*kernel*

[in] Хендл на кернел OpenCL.

*work\_dim*

[in] Размерность пространства задач.

*global\_work\_offset[]*

[in] Начальное смещение в пространстве задач.

*global\_work\_size[]*

[in] Размер подмножества задач.

*local\_work\_size[]*

[in] Размер локального подмножества задач в группе.

### Возвращаемое значение

Возвращает true при успешном выполнении, в противном случае false. Для получения информации об ошибке используйте функцию [GetLastError\(\)](#).

### Примечание

Рассмотрим на примере смысл параметров:

- *work\_dim* задает размерность массива *work\_items[]*, описывающего задачи. Если *work\_dim*=3, то используется 3-мерный массив *work\_items[N1, N2, N3]*.
- *global\_work\_size[]* содержит значения, задающие размер массива *work\_items[]*. Если у нас *work\_dim*=3, и соответственно, массив *global\_work\_size[3]* может быть {40, 100, 320}. Тогда имеем *work\_items[40, 100, 320]*. Значит общее количество задач равно  $40 \times 100 \times 320 = 1\,280\,000$ .
- *local\_work\_size[]* задает подмножество задач, которые будут выполняться указанным кернелом OpenCL программы. Его размерность равна размерности *work\_items[]* и позволяет общее подмножество задач нарезать на более мелкие подмножества без остатков от деления. Фактически, размеры массива *local\_work\_size[]* должны быть подобраны таким образом, чтобы глобальное множество задач *work\_items[]* нарезалось на более мелкие подмножества. В данном примере подойдет *local\_work\_size[3]={10, 10, 10}*, так как *work\_items[40, 100, 320]* без остатка можно собрать из массива *local\_items[10, 10, 10]*.

## CLExecutionStatus

Возвращает состояние выполнения OpenCL программы.

```
int CLExecutionStatus(
    int kernel           // хендл на кернел OpenCL программы
);
```

### Параметры

*kernel*

[in] Хендл на кернел программы OpenCL.

### Возвращаемое значение

Возвращает статус OpenCL программы, значение может быть одним из нижеследующих:

- CL\_COMPLETE=0 - программа завершена,
- CL\_RUNNING=1 - выполняется,
- CL\_SUBMITTED=2 - отправлена на выполнение,
- CL\_QUEUED=3 - находится в очереди на выполнение,
- -1 (минус один) - произошла ошибка при выполнении CLExecutionStatus().

## Интеграция MetaTrader со сторонними программами

MQL5 предназначен для разработки высокопроизводительных торговых приложений на финансовых рынках и не имеет аналогов среди других специализированных языков, используемых в алготрейдинге. Синтаксис и скорость работы программ на MQL5 максимально приближены к C++, имеется поддержка [OpenCL](#) и [интеграция с MS Visual Studio](#), доступны библиотеки [статистики](#), [нечеткой логики](#) и [ALGLIB](#). В среде разработки MetaEditor также имеется нативная [поддержка .NET библиотек](#) с "умным" импортом функций без написания специальных оберток, можно использовать сторонние C++ DLL. Файлы исходных кодов на C++ (CPP и H) можно редактировать и компилировать в DLL прямо из редактора — для этого используется Microsoft Visual Studio, установленный на компьютере пользователя.

В дополнение ко всем вышеперечисленным преимуществам платформа MetaTrader 5 предоставляет разработчикам на MQL5 возможность интеграции с другими наиболее популярными решениями для обработки финансовых данных. Это позволяет специалистам по статистике и машинному обучению получать данные прямо из MetaTrader 5 без написания дополнительных программ и адаптеров.

В данном разделе описываются следующие решения по интеграции MetaTrader

- [MetaTrader для Python](#) - модуль для работы с программами на Python.

Представленные функции позволяют подключаться к терминалу MetaTrader 5 напрямую и запрашивать ценовую историю в необходимом объеме и виде для любых финансовых инструментов, доступных в платформе.

## Модуль MetaTrader для интеграции с Python

Python является современным высокоуровневым языком программирования для разработки сценариев и приложений. Содержит множество библиотек для машинного обучения, автоматизации процессов, анализа и визуализации данных.

Пакет MetaTrader для Python предназначен для удобного и быстрого получения биржевой информации через межпроцессное взаимодействие прямо из терминала MetaTrader 5. Полученные таким образом данные можно дальше использовать для статистических вычислений и машинного обучения.

Функции для интеграции MetaTrader 5 и Python

Функция	Действие
<a href="#">MT5Initialize</a>	Устанавливает соединение с терминалом MetaTrader 5
<a href="#">MT5Shutdown</a>	Закрывает ранее установленное подключение к терминалу MetaTrader 5
<a href="#">MT5TerminalInfo</a>	Получает состояние и параметры подключенного терминала MetaTrader 5
<a href="#">MT5Version</a>	Возвращает версию терминала MetaTrader 5
<a href="#">MT5WaitForTerminal</a>	Ждет пока терминал MetaTrader 5 подключится к торговому серверу
<a href="#">MT5CopyRatesFrom</a>	Получает бары из терминала MetaTrader 5, начиная с указанной даты
<a href="#">MT5CopyRatesFromPos</a>	Получает бары из терминала MetaTrader 5, начиная с указанного индекса
<a href="#">MT5CopyRatesRange</a>	Получает бары в указанном диапазоне дат из терминала MetaTrader 5
<a href="#">MT5CopyTicksFrom</a>	Получает тики из терминала MetaTrader 5, начиная с указанной даты
<a href="#">MT5CopyTicksRange</a>	Получает тики за указанный диапазон дат из терминала MetaTrader 5

### Пример подключения Python к MetaTrader 5

- Скачайте последнюю версию Python 3.7 со страницы <https://www.python.org/downloads/windows>
- При установке Python отметьте чек-бокс "Add Python 3.7 to PATH%", чтобы можно было из командной строки запускать скрипты на Python.
- Установите модуль MetaTrader5 из командной строки  

```
pip install MetaTrader5
```
- Добавьте пакеты matplotlib и pytz

```
pip install matplotlib
pip install pytz
```

## 5. Запустите тестовый скрипт

```
from datetime import datetime
from MetaTrader5 import *
from pytz import timezone
import matplotlib.pyplot as plt
utc_tz = timezone('UTC')

# подключимся к MetaTrader 5
MT5Initialize()
# подождем, пока терминал MetaTrader 5 установит соединение с торговым сервером и синхронизируется
MT5WaitForTerminal()

# запросим статус и параметры подключения
print(MT5TerminalInfo())
# получим информацию о версии MetaTrader 5
print(MT5Version())

# запросим 1000 тиков с EURAUD
euraud_ticks = MT5CopyTicksFrom("EURAUD", datetime(2019,4,1,0), 1000, MT5_COPY_TICKS_ASK)
# запросим тики с AUDUSD в интервале 2019.04.01 13:00 – 2019.04.02 13:00
audusd_ticks = MT5CopyTicksRange("AUDUSD", datetime(2019,4,1,13), datetime(2019,4,2,13))

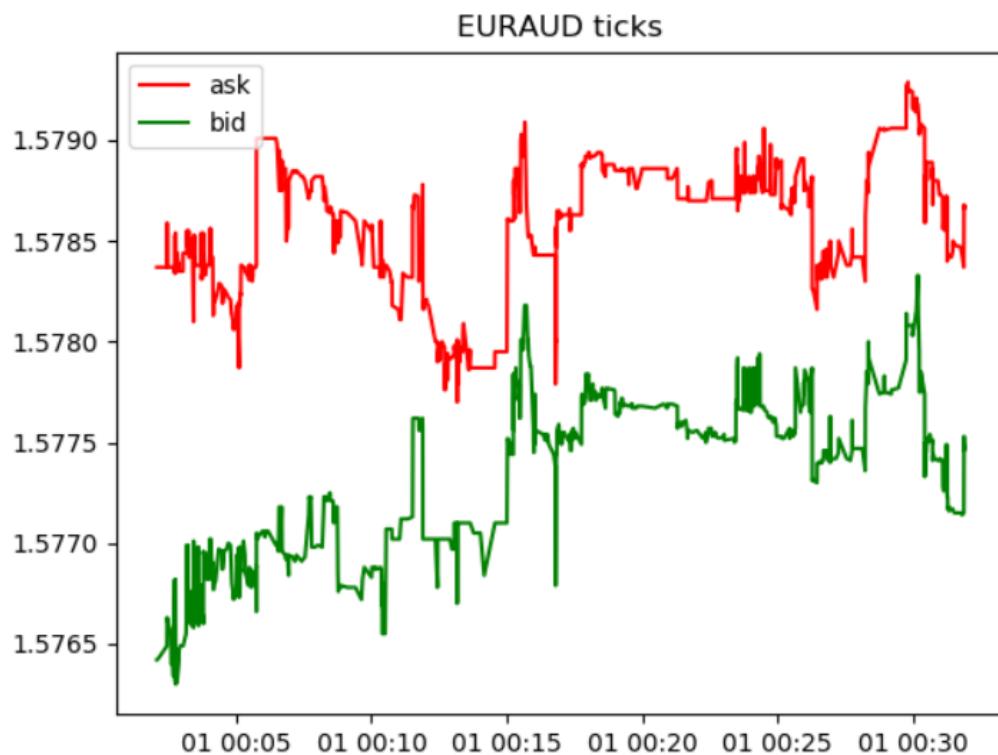
# получим бары с разных инструментов разными способами
eurusd_rates = MT5CopyRatesFrom("EURUSD", MT5_TIMEFRAME_M1, datetime(2019,4,5,15), 1000)
eurrub_rates = MT5CopyRatesFromPos("EURRUB", MT5_TIMEFRAME_M1, 0, 1000)
eurjpy_rates = MT5CopyRatesRange("EURJPY", MT5_TIMEFRAME_M1, datetime(2019,4,1,13), datetime(2019,4,2,13))
# завершим подключение к MetaTrader 5
MT5Shutdown()

#DATA
print('euraud_ticks(', len(euraud_ticks), ')')
for val in euraud_ticks[:10]: print(val)
print('audusd_ticks(', len(audusd_ticks), ')')
for val in audusd_ticks[:10]: print(val)
print('eurusd_rates(', len(eurusd_rates), ')')
for val in eurusd_rates[:10]: print(val)
print('eurrub_rates(', len(eurrub_rates), ')')
for val in eurrub_rates[:10]: print(val)
print('eurjpy_rates(', len(eurjpy_rates), ')')
for val in eurjpy_rates[:10]: print(val)

#PLOTTING
x_time = [x.time.astimezone(utc_tz) for x in euraud_ticks]
# подготовим массивы Bid и Ask
bid = [y.bid for y in euraud_ticks]
ask = [y.ask for y in euraud_ticks]
```

```
# сделаем отрисовку тиков на графике
plt.plot(x_time, ask,'r-', label='ask')
plt.plot(x_time, bid,'g-', label='bid')
# выведем легенды
plt.legend(loc='upper left')
# добавим заголовок
plt.title('EURAUD ticks')
# покажем график
plt.show()
```

6. Получите данные и график



```
[2, 'MetaQuotes-Demo', '16167573']
[500, 2009, '15 Mar 2019']

euraud_ticks( 1000 )
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 3, 512000), bid=1.5764200000000002, a
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 8, 70000), bid=1.57643, ask=1.57837,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 26, 142000), bid=1.57649, ask=1.57837
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 26, 260000), bid=1.5765500000000001,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 26, 365000), bid=1.5765500000000001,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 26, 410000), bid=1.5765500000000001,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 26, 636000), bid=1.57663, ask=1.57837
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 30, 72000), bid=1.57659, ask=1.57837,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 34, 320000), bid=1.57656, ask=1.57837
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 35, 61000), bid=1.57653, ask=1.57837,
```

```

audusd_ticks( 61336 )
MT5Tick(time=datetime.datetime(2019, 4, 1, 13, 0, 1, 410000), bid=0.71262, ask=0.7128,
MT5Tick(time=datetime.datetime(2019, 4, 1, 13, 0, 2, 380000), bid=0.71263, ask=0.71282
MT5Tick(time=datetime.datetime(2019, 4, 1, 13, 0, 2, 753000), bid=0.71262, ask=0.71282
MT5Tick(time=datetime.datetime(2019, 4, 1, 13, 0, 2, 842000), bid=0.71262, ask=0.71282
MT5Tick(time=datetime.datetime(2019, 4, 1, 13, 0, 3, 428000), bid=0.71261, ask=0.71279
MT5Tick(time=datetime.datetime(2019, 4, 1, 13, 0, 3, 771000), bid=0.71261, ask=0.7128,
MT5Tick(time=datetime.datetime(2019, 4, 1, 13, 0, 3, 936000), bid=0.71261, ask=0.71279
MT5Tick(time=datetime.datetime(2019, 4, 1, 13, 0, 4, 2000), bid=0.7126, ask=0.71279,
MT5Tick(time=datetime.datetime(2019, 4, 1, 13, 0, 4, 108000), bid=0.71258, ask=0.71276
MT5Tick(time=datetime.datetime(2019, 4, 1, 13, 0, 4, 186000), bid=0.71257, ask=0.71275

eurusd_rates( 1000 )
MT5Rate(time=datetime.datetime(2019, 4, 4, 22, 20), open=1.12162, low=1.12171, high=1.
MT5Rate(time=datetime.datetime(2019, 4, 4, 22, 21), open=1.12153, low=1.1217, high=1.
MT5Rate(time=datetime.datetime(2019, 4, 4, 22, 22), open=1.1217, low=1.12179, high=1.
MT5Rate(time=datetime.datetime(2019, 4, 4, 22, 23), open=1.12178, low=1.12189, high=1.
MT5Rate(time=datetime.datetime(2019, 4, 4, 22, 24), open=1.12188, low=1.12188, high=1.
MT5Rate(time=datetime.datetime(2019, 4, 4, 22, 25), open=1.12183, low=1.12187, high=1.
MT5Rate(time=datetime.datetime(2019, 4, 4, 22, 26), open=1.12174, low=1.12179, high=1.
MT5Rate(time=datetime.datetime(2019, 4, 4, 22, 27), open=1.12178, low=1.12179, high=1.
MT5Rate(time=datetime.datetime(2019, 4, 4, 22, 28), open=1.12171, low=1.12171, high=1.
MT5Rate(time=datetime.datetime(2019, 4, 4, 22, 29), open=1.12169, low=1.12169, high=1.

eurrub_rates( 1000 )
MT5Rate(time=datetime.datetime(2019, 4, 3, 18, 49), open=73.369, low=73.371, high=73.3
MT5Rate(time=datetime.datetime(2019, 4, 3, 18, 50), open=73.353, low=73.367, high=73.3
MT5Rate(time=datetime.datetime(2019, 4, 3, 18, 51), open=73.367, low=73.367, high=73.3
MT5Rate(time=datetime.datetime(2019, 4, 3, 18, 52), open=73.356, low=73.358, high=73.3
MT5Rate(time=datetime.datetime(2019, 4, 3, 18, 53), open=73.347, low=73.35, high=73.3
MT5Rate(time=datetime.datetime(2019, 4, 3, 18, 54), open=73.333, low=73.337, high=73.3
MT5Rate(time=datetime.datetime(2019, 4, 3, 18, 55), open=73.32, low=73.33, high=73.32,
MT5Rate(time=datetime.datetime(2019, 4, 3, 18, 56), open=73.328, low=73.334, high=73.3
MT5Rate(time=datetime.datetime(2019, 4, 3, 18, 57), open=73.326, low=73.327, high=73.3
MT5Rate(time=datetime.datetime(2019, 4, 3, 18, 58), open=73.32, low=73.331, high=73.3

eurjpy_rates( 1441 )
MT5Rate(time=datetime.datetime(2019, 4, 1, 13, 0), open=124.763, low=124.785, high=124
MT5Rate(time=datetime.datetime(2019, 4, 1, 13, 1), open=124.748, low=124.754, high=124
MT5Rate(time=datetime.datetime(2019, 4, 1, 13, 2), open=124.752, low=124.768, high=124
MT5Rate(time=datetime.datetime(2019, 4, 1, 13, 3), open=124.752, low=124.756, high=124
MT5Rate(time=datetime.datetime(2019, 4, 1, 13, 4), open=124.738, low=124.76, high=124
MT5Rate(time=datetime.datetime(2019, 4, 1, 13, 5), open=124.756, low=124.773, high=124
MT5Rate(time=datetime.datetime(2019, 4, 1, 13, 6), open=124.773, low=124.802, high=124
MT5Rate(time=datetime.datetime(2019, 4, 1, 13, 7), open=124.777, low=124.781, high=124
MT5Rate(time=datetime.datetime(2019, 4, 1, 13, 8), open=124.776, low=124.79, high=124
MT5Rate(time=datetime.datetime(2019, 4, 1, 13, 9), open=124.777, low=124.803, high=124

```



## MT5Initialize

Устанавливает соединение с терминалом MetaTrader 5.

```
MT5Initialize(
    path=None          // путь к EXE-файлу терминала MetaTrader 5
)
```

### Параметры

*path=None*

[in] Путь к файлу metatrader.exe или metatrader64.exe. Если путь не указан, модуль попытается найти исполняемый файл самостоятельно.

### Возвращаемое значение

Возвращает True в случае успешного подключения к терминалу MetaTrader 5, иначе - False.

### Примечание

Если потребуется, при выполнении вызова MT5Initialize() терминал MetaTrader 5 будет запущен для выполнения соединения.

### Пример:

```
from datetime import datetime
from MetaTrader5 import *

# установим подключение к терминалу MetaTrader 5
MT5Initialize()

# подождем пока MetaTrader 5 подключится к торговому серверу
MT5WaitForTerminal()

# выведем информацию о состоянии подключения, названии сервера и торговом счете
print(MT5TerminalInfo())

# выведем информацию о версии MetaTrader 5
print(MT5Version())

# завершим подключение к терминалу MetaTrader 5
MT5Shutdown()
```

### Смотри также

[MT5Shutdown](#), [MT5WaitForTerminal](#), [MT5TerminalInfo](#), [MT5Version](#)

## MT5Shutdown

Закрывает ранее установленное подключение к терминалу MetaTrader 5.

```
MT5Shutdown()
```

### Возвращаемое значение

Нет.

### Пример:

```
from datetime import datetime
from MetaTrader5 import *

# установим подключение к терминалу MetaTrader 5
MT5Initialize()
# подождем пока MetaTrader 5 подключится к торговому серверу
MT5WaitForTerminal()
# выведем информацию о состоянии подключения, названии сервера и торговом счете
print(MT5TerminalInfo())
# выведем информацию о версии MetaTrader 5
print(MT5Version())
# завершим подключение к терминалу MetaTrader 5
MT5Shutdown()
```

### Смотри также

[MT5Initialize](#), [MT5WaitForTerminal](#), [MT5TerminalInfo](#), [MT5Version](#)

## MT5TerminalInfo

Получает состояние и параметры подключенного терминала MetaTrader 5.

```
MT5TerminalInfo()
```

### Возвращаемое значение

Возвращает состояние подключения, имя торгового сервера и номер торгового счета.

### Примечание

Значение возвращается в виде кортежа из трех значений:

Тип	Описание	Пример значения
integer	Статус подключения к торговому серверу: 0 - подключения нет, 1 - терминал подключился, но торговое окружение еще не синхронизировано, 2 - терминал подключен к торговому серверу	2
string	Имя торгового сервера	'MetaQuotes-Demo'
string	Номер торгового счета (логин)	'15185779'

### Пример:

```
from datetime import datetime
from MetaTrader5 import *

# установим подключение к терминалу MetaTrader 5
MT5Initialize()

# подождем пока MetaTrader 5 подключится к торговому серверу
MT5WaitForTerminal()

# выведем информацию о состоянии подключения, названии сервера и торговом счете
print(MT5TerminalInfo())

# выведем информацию о версии MetaTrader 5
print(MT5Version())

# завершим подключение к терминалу MetaTrader 5
MT5Shutdown()

Результат:
>>> from datetime import datetime
>>> from MetaTrader5 import *
>>> MT5Initialize()
True
>>> MT5WaitForTerminal()
True
>>> print(MT5TerminalInfo())
```

```
[2, 'MetaQuotes-Demo', '16167573']
>>> print(MT5Version())
[500, 2009, '15 Mar 2019']
>>> MT5Shutdown()
True
```

**Смотри также**

[MT5Initialize](#), [MT5WaitForTerminal](#), [MT5Shutdown](#), [MT5Version](#)

## MT5Version

Возвращает версию терминала MetaTrader 5.

```
MT5Version()
```

### Возвращаемое значение

Возвращает номер версии, номер билда и дату релиза терминала MetaTrader 5.

### Примечание

Функция MT5Version() возвращает состояние подключения, адрес торгового сервера и номер торгового счета в виде кортежа:

### Примечание

Значение возвращается в виде кортежа из трех значений:

Тип	Описание	Пример значения
integer	Версия терминала MetaTrader 5	500
string	Номер билда	'2007'
string	Дата релиза билда	'25 Feb 2019'

### Пример:

```
from datetime import datetime
from MetaTrader5 import *

# установим подключение к терминалу MetaTrader 5
MT5Initialize()

# подождем пока MetaTrader 5 подключится к торговому серверу
MT5WaitForTerminal()

# выведем информацию о состоянии подключения, названии сервера и торговом счете
print(MT5TerminalInfo())

# выведем информацию о версии MetaTrader 5
print(MT5Version())

# завершим подключение к терминалу MetaTrader 5
MT5Shutdown()

Результат:
>>> from datetime import datetime
>>> from MetaTrader5 import *
>>> MT5Initialize()
True
>>> MT5WaitForTerminal()
True
>>> print(MT5TerminalInfo())
[2, 'MetaQuotes-Demo', '16167573']
>>> print(MT5Version())
```

```
[500, 2009, '15 Mar 2019']
>>> MT5Shutdown()
True
```

#### Смотри также

[MT5Initialize](#), [MT5WaitForTerminal](#), [MT5Shutdown](#), [MT5TerminalInfo](#)

## MT5WaitForTerminal

Ждет пока терминал MetaTrader 5 подключится к торговому серверу.

```
MT5WaitForTerminal()
```

### Возвращаемое значение

Возвращает True в случае успешного подключения терминала MetaTrader 5 к торговому серверу, иначе - False.

### Примечание

Функция MT5Version() возвращает состояние подключения, адрес торгового сервера и номер торгового счета в виде кортежа:

### Примечание

Время ожидания составляет 1 минуту. Для проверки состояния подключения после MT5WaitForTerminal() необходимо вызвать [MT5TerminalInfo\(\)](#).

### Пример:

```
from datetime import datetime
from MetaTrader5 import *

# установим подключение к терминалу MetaTrader 5
MT5Initialize()
# подождем пока MetaTrader 5 подключится к торговому серверу
MT5WaitForTerminal()
# выведем информацию о состоянии подключения, названии сервера и торговом счете
print(MT5TerminalInfo())
# выведем информацию о версии MetaTrader 5
print(MT5Version())
# завершим подключение к терминалу MetaTrader 5
MT5Shutdown()
```

### Смотри также

[MT5Initialize](#), [MT5TerminalInfo](#), [MT5Version](#), [MT5Shutdown](#)

## MT5CopyRatesFrom

Получает бары из терминала MetaTrader 5, начиная с указанной даты.

```
MT5CopyRatesFrom(
    symbol,           // имя символа
    timeframe,        // таймфрейм
    from,             // дата открытия начального бара
    count            // количество баров
)
```

### Параметры

*symbol*

[in] Имя финансового инструмента, например, "EURUSD".

*timeframe*

[in] Таймфрейм, для которого запрашиваются бары. Задается значением из перечисления [MT5\\_TIMEFRAME](#).

*from*

[in] Дата открытия первого бара из запрашиваемой выборки. Задается объектом `datetime` или в виде количества секунд, прошедших с 1970.01.01.

*count*

[in] Количество баров, которое необходимо получить.

### Возвращаемое значение

Возвращает бары в виде кортежей (`time, open, high, low, close, tick_volume, spread, real_volume`).

### Примечание

Для дополнительной информации смотрите функцию [CopyRates\(\)](#).

Python при создании объекта `datetime` использует таймзону локального времени, в то время как терминал MetaTrader 5 хранит время тиков и открытия баров в UTC таймзоне (без смещения). Поэтому, для выполнения функций, использующих время, необходимо создавать `datetime` в UTC-времени. Данные, полученные из терминала MetaTrader 5, имеют UTC-время, но при попытке вывести их на печать, Python опять применит смещение для локального времени. Поэтому, полученные данные также необходимо корректировать для визуального представления.

`MT5_TIMEFRAME` является перечислением с возможными значениями периодов графика

Идентификатор	Описание
MT5_TIMEFRAME_M1	1 минута
MT5_TIMEFRAME_M2	2 минуты
MT5_TIMEFRAME_M3	3 минуты
MT5_TIMEFRAME_M4	4 минуты

MT5_TIMEFRAME_M5	5 минут
MT5_TIMEFRAME_M6	6 минут
MT5_TIMEFRAME_M10	10 минут
MT5_TIMEFRAME_M12	12 минут
MT5_TIMEFRAME_M12	15 минут
MT5_TIMEFRAME_M20	20 минут
MT5_TIMEFRAME_M30	30 минут
MT5_TIMEFRAME_H1	1 час
MT5_TIMEFRAME_H2	2 часа
MT5_TIMEFRAME_H3	3 часа
MT5_TIMEFRAME_H4	4 часа
MT5_TIMEFRAME_H6	6 часов
MT5_TIMEFRAME_H8	8 часов
MT5_TIMEFRAME_H12	12 часов
MT5_TIMEFRAME_D1	1 день
MT5_TIMEFRAME_W1	1 неделя
MT5_TIMEFRAME_MON1	1 месяц

Пример:

```
from datetime import datetime
from MetaTrader5 import *

# импортируем модуль pandas для вывода полученных данных в табличной форме
import pandas as pd
pd.set_option('display.max_columns', 500) # сколько столбцов показываем
pd.set_option('display.width', 1500)       # макс. ширина таблицы для показа
# импортируем модуль pytz для работы с таймзоной
import pytz

# установим подключение к терминалу MetaTrader 5
MT5Initialize()
# подождем пока MetaTrader 5 подключится к торговому серверу
MT5WaitForTerminal()

# установим таймзону в UTC
timezone = pytz.timezone("Etc/UTC")
# создадим объект datetime в таймзоне UTC, чтобы не применялось смещение локальной таймы
utc_from = datetime(2019, 4, 5, tzinfo=timezone)
```

```

# получим 10 баров с EURUSD H4 начиная с 01.04.2019 в таймзоне UTC
rates = MT5CopyRatesFrom("EURUSD", MT5_TIMEFRAME_H4, utc_from, 10)

# завершим подключение к терминалу MetaTrader 5
MT5Shutdown()

# выведем каждый элемент полученных данных на новой строке
print("Выведем полученные данные как есть")
for rate in rates:
    print(rate)

# создадим из полученных данных DataFrame
rates_frame = pd.DataFrame(list(rates),
                            columns=['time', 'open', 'low', 'high', 'close', 'tick_volume'])

# выведем данные
print("\nВыведем датафрейм с данными")
print(rates_frame) # видим, что Python представляет время открытия баров в локальной

# получим для локального компьютера смещение от времени UTC
UTC_OFFSET_TIMDELTA = datetime.utcnow() - datetime.now()

# создадим простую функцию, которая влоб корректирует смещение
def local_to_utc(dt):
    return dt + UTC_OFFSET_TIMDELTA

# применим смещение для столбца time в датафрейме rates_frame
rates_frame['time'] = rates_frame.apply(lambda rate: local_to_utc(rate['time']), axis=1)

# еще раз выведем данные и убедимся, что теперь время открытия 4-часовых свечей кратно 4 часам
print("\nВыведем датафрейм после корректировки времени")
print(rates_frame)

```

Результат:

Выведем полученные данные как есть

```

MT5Rate(time=datetime.datetime(2019, 4, 3, 15, 0), open=1.12431, low=1.12543, high=1.12559, close=1.12341, tick_volume=11735, spread=8)
MT5Rate(time=datetime.datetime(2019, 4, 3, 19, 0), open=1.12339, low=1.12487, high=1.12559, close=1.12439, tick_volume=15241, spread=8)
MT5Rate(time=datetime.datetime(2019, 4, 3, 23, 0), open=1.12439, low=1.12478, high=1.12559, close=1.12311, tick_volume=4973, spread=8)
MT5Rate(time=datetime.datetime(2019, 4, 4, 3, 0), open=1.12324, low=1.12472, high=1.12559, close=1.12218, tick_volume=11735, spread=8)
MT5Rate(time=datetime.datetime(2019, 4, 4, 7, 0), open=1.1244, low=1.12451, high=1.12559, close=1.12311, tick_volume=11735, spread=8)
MT5Rate(time=datetime.datetime(2019, 4, 4, 11, 0), open=1.12406, low=1.12472, high=1.12559, close=1.12218, tick_volume=11735, spread=8)
MT5Rate(time=datetime.datetime(2019, 4, 4, 15, 0), open=1.12313, low=1.12359, high=1.12559, close=1.12218, tick_volume=11735, spread=8)
MT5Rate(time=datetime.datetime(2019, 4, 4, 19, 0), open=1.12171, low=1.12246, high=1.12559, close=1.12218, tick_volume=11735, spread=8)
MT5Rate(time=datetime.datetime(2019, 4, 4, 23, 0), open=1.12187, low=1.12265, high=1.12559, close=1.12218, tick_volume=11735, spread=8)
MT5Rate(time=datetime.datetime(2019, 4, 5, 3, 0), open=1.12199, low=1.12286, high=1.12559, close=1.12218, tick_volume=11735, spread=8)

```

Выведем датафрейм с данными

	time	open	low	high	close	tick_volume	spread	real_volume
0	2019-04-03 15:00:00	1.12431	1.12543	1.12335	1.12341	11735	8	
1	2019-04-03 19:00:00	1.12339	1.12487	1.12247	1.12439	15241	8	
2	2019-04-03 23:00:00	1.12439	1.12478	1.12311	1.12312	4973	8	

3	2019-04-04	03:00:00	1.12324	1.12472	1.12318	1.12440	3099	4
4	2019-04-04	07:00:00	1.12440	1.12451	1.12364	1.12406	3304	4
5	2019-04-04	11:00:00	1.12406	1.12472	1.12302	1.12313	10119	8
6	2019-04-04	15:00:00	1.12313	1.12359	1.12122	1.12171	15098	8
7	2019-04-04	19:00:00	1.12171	1.12246	1.12056	1.12188	15369	8
8	2019-04-04	23:00:00	1.12187	1.12265	1.12130	1.12191	5156	8
9	2019-04-05	03:00:00	1.12200	1.12286	1.12170	1.12220	3903	4

Выведем датафрейм после корректировки времени открытия свечей

	time	open	low	high	close	tick_volume	spread	real_v
0	2019-04-03 12:00:00	1.12431	1.12543	1.12335	1.12341	11735	8	
1	2019-04-03 16:00:00	1.12339	1.12487	1.12247	1.12439	15241	8	
2	2019-04-03 20:00:00	1.12439	1.12478	1.12311	1.12312	4973	8	
3	2019-04-04 00:00:00	1.12324	1.12472	1.12318	1.12440	3099	4	
4	2019-04-04 04:00:00	1.12440	1.12451	1.12364	1.12406	3304	4	
5	2019-04-04 08:00:00	1.12406	1.12472	1.12302	1.12313	10119	8	
6	2019-04-04 12:00:00	1.12313	1.12359	1.12122	1.12171	15098	8	
7	2019-04-04 16:00:00	1.12171	1.12246	1.12056	1.12188	15369	8	
8	2019-04-04 20:00:00	1.12187	1.12265	1.12130	1.12191	5156	8	
9	2019-04-05 00:00:00	1.12200	1.12286	1.12170	1.12220	3903	4	

#### Смотри также

[CopyRates](#), [MT5CopyRatesFromPos](#), [MT5CopyRatesRange](#), [MT5CopyTicksFrom](#), [MT5CopyTicksRange](#)

## MT5CopyRatesFromPos

Получает бары из терминала MetaTrader 5, начиная с указанного индекса.

```
MT5CopyRatesFromPos(
    symbol,           // имя символа
    timeframe,        // таймфрейм
    start_pos,        // номер начального бара
    count             // количество баров
)
```

### Параметры

*symbol*

[in] Имя финансового инструмента, например, "EURUSD".

*timeframe*

[in] Таймфрейм, для которого запрашиваются бары. Задается значением из перечисления [MT5\\_TIMEFRAME](#).

*start\_pos*

[in] Начальный номер бара, с которого запрашиваются данные. Нумерация баров идет от настоящего к прошлому, то есть нулевой бар означает текущий.

*count*

[in] Количество баров, которое необходимо получить.

### Возвращаемое значение

Возвращает бары в виде кортежей (time, open, high, low, close, tick\_volume, spread, real\_volume).

### Примечание

Для дополнительной информации смотрите функцию [CopyRates\(\)](#).

### Пример:

```
from datetime import datetime
from MetaTrader5 import *
# импортируем модуль pandas для вывода полученных данных в табличной форме
import pandas as pd
pd.set_option('display.max_columns', 500) # сколько столбцов показываем
pd.set_option('display.width', 1500)       # макс. ширина таблицы для показа

# установим подключение к терминалу MetaTrader 5
MT5Initialize()
# подождем пока MetaTrader 5 подключится к торговому серверу
MT5WaitForTerminal()

# запросим 10 баров на GBPUSD D1 с текущего дня
rates = MT5CopyRatesFrom("GBPUSD", MT5_TIMEFRAME_D1, 0, 10)
```

```

# завершим подключение к терминалу MetaTrader 5
MT5Shutdown()

# выведем каждый элемент полученных данных на новой строке
print("Выведем полученные данные как есть")
for rate in rates:
    print(rate)

# создадим из полученных данных DataFrame
rates_frame = pd.DataFrame(list(rates),
                            columns=['time', 'open', 'low', 'high', 'close', 'tick_volume',
                            'spread', 'real_volume'])

# выведем данные
print("\nВыведем датафрейм с данными")
print(rates_frame) # видим, что Python представляет время открытия баров в локальной

# получим для локального компьютера смещение от времени UTC
UTC_OFFSET_TIMDELTA = datetime.utcnow() - datetime.now()

# создадим простую функцию, которая влоб корректирует смещение
def local_to_utc(dt):
    return dt + UTC_OFFSET_TIMDELTA

# применим смещение для столбца time в датафрейме rates_frame
rates_frame['time'] = rates_frame.apply(lambda rate: local_to_utc(rate['time']), axis=1)

# еще раз выведем данные и убедимся, что теперь время открытия дневных свечей равно 00:00
print("\nВыведем датафрейм после корректировки времени открытия свечей ")
print(rates_frame)

```

Результат:

Выведем полученные данные как есть

```

MT5Rate(time=datetime.datetime(2019, 4, 3, 15, 0), open=1.12431, low=1.12543, high=1.12559, spread=8, real_volume=11735)
MT5Rate(time=datetime.datetime(2019, 4, 3, 19, 0), open=1.12339, low=1.12487, high=1.12559, spread=8, real_volume=15241)
MT5Rate(time=datetime.datetime(2019, 4, 3, 23, 0), open=1.12439, low=1.12478, high=1.12559, spread=8, real_volume=4973)
MT5Rate(time=datetime.datetime(2019, 4, 4, 3, 0), open=1.12324, low=1.12472, high=1.12559, spread=4, real_volume=3099)
MT5Rate(time=datetime.datetime(2019, 4, 4, 7, 0), open=1.1244, low=1.12451, high=1.12559, spread=4, real_volume=3304)
MT5Rate(time=datetime.datetime(2019, 4, 4, 11, 0), open=1.12406, low=1.12472, high=1.12559, spread=4, real_volume=10119)
MT5Rate(time=datetime.datetime(2019, 4, 4, 15, 0), open=1.12313, low=1.12359, high=1.12559, spread=4, real_volume=8)
MT5Rate(time=datetime.datetime(2019, 4, 4, 19, 0), open=1.12171, low=1.12246, high=1.12559, spread=4, real_volume=8)
MT5Rate(time=datetime.datetime(2019, 4, 4, 23, 0), open=1.12187, low=1.12265, high=1.12559, spread=4, real_volume=8)
MT5Rate(time=datetime.datetime(2019, 4, 5, 3, 0), open=1.12199, low=1.12286, high=1.12559, spread=8, real_volume=8)

```

Выведем датафрейм с данными

	time	open	low	high	close	tick_volume	spread	real_volume
0	2019-04-03 15:00:00	1.12431	1.12543	1.12335	1.12341	11735	8	
1	2019-04-03 19:00:00	1.12339	1.12487	1.12247	1.12439	15241	8	
2	2019-04-03 23:00:00	1.12439	1.12478	1.12311	1.12312	4973	8	
3	2019-04-04 03:00:00	1.12324	1.12472	1.12318	1.12440	3099	4	
4	2019-04-04 07:00:00	1.12440	1.12451	1.12364	1.12406	3304	4	
5	2019-04-04 11:00:00	1.12406	1.12472	1.12302	1.12313	10119	8	

6	2019-04-04	15:00:00	1.12313	1.12359	1.12122	1.12171	15098	8
7	2019-04-04	19:00:00	1.12171	1.12246	1.12056	1.12188	15369	8
8	2019-04-04	23:00:00	1.12187	1.12265	1.12130	1.12191	5156	8
9	2019-04-05	03:00:00	1.12200	1.12286	1.12170	1.12220	3903	4

Выведем датафрейм после корректировки времени открытия свечей

	time	open	low	high	close	tick_volume	spread	real_volume
0	2019-04-03 12:00:00	1.12431	1.12543	1.12335	1.12341	11735	8	
1	2019-04-03 16:00:00	1.12339	1.12487	1.12247	1.12439	15241	8	
2	2019-04-03 20:00:00	1.12439	1.12478	1.12311	1.12312	4973	8	
3	2019-04-04 00:00:00	1.12324	1.12472	1.12318	1.12440	3099	4	
4	2019-04-04 04:00:00	1.12440	1.12451	1.12364	1.12406	3304	4	
5	2019-04-04 08:00:00	1.12406	1.12472	1.12302	1.12313	10119	8	
6	2019-04-04 12:00:00	1.12313	1.12359	1.12122	1.12171	15098	8	
7	2019-04-04 16:00:00	1.12171	1.12246	1.12056	1.12188	15369	8	
8	2019-04-04 20:00:00	1.12187	1.12265	1.12130	1.12191	5156	8	
9	2019-04-05 00:00:00	1.12200	1.12286	1.12170	1.12220	3903	4	

#### Смотри также

[CopyRates](#), [MT5CopyRatesFrom](#), [MT5CopyRatesRange](#), [MT5CopyTicksFrom](#), [MT5CopyTicksRange](#)

## MT5CopyRatesRange

Получает бары в указанном диапазоне дат из терминала MetaTrader 5.

```
MT5CopyRatesRange (
    symbol,           // имя символа
    timeframe,        // таймфрейм
    date_from,        // дата, с которой запрашиваются бары
    date_to          // дата, по которую запрашиваются бары
)
```

### Параметры

*symbol*

[in] Имя финансового инструмента, например, "EURUSD".

*timeframe*

[in] Таймфрейм, для которого запрашиваются бары. Задается значением из перечисления [MT5\\_TIMEFRAME](#).

*date\_from*

[in] Дата, начиная с которой запрашиваются бары. Задается объектом `datetime` или в виде количества секунд, прошедших с 1970.01.01. Отдаются бары со временем открытия  $\geq$  *date\_from*.

*date\_to*

[in] Дата, по которую запрашиваются бары. Задается объектом `datetime` или в виде количества секунд, прошедших с 1970.01.01. Отдаются бары со временем открытия  $\leq$  *date\_to*.

### Возвращаемое значение

Возвращает бары в виде кортежей (`time, open, high, low, close, tick_volume, spread, real_volume`).

### Примечание

Для дополнительной информации смотрите функцию [CopyRates\(\)](#).

Python при создании объекта `datetime` использует таймзону локального времени, в то время как терминал MetaTrader 5 хранит время тиков и открытия баров в UTC таймзоне (без смещения). Поэтому, для выполнения функций, использующих время, необходимо создавать `datetime` в UTC-времени. Данные, полученные из терминала MetaTrader 5, имеют UTC-время, но при попытке вывести их на печать, Python опять применит смещение для локального времени. Поэтому, полученные данные также необходимо корректировать для визуального представления.

### Пример:

```
from datetime import datetime
from MetaTrader5 import *
# импортируем модуль pandas для вывода полученных данных в табличной форме
import pandas as pd
pd.set_option('display.max_columns', 500) # сколько столбцов показываем
```

```

pd.set_option('display.width', 1500)      # макс. ширина таблицы для показа

# установим подключение к терминалу MetaTrader 5
MT5Initialize()
# подождем пока MetaTrader 5 подключится к торговому серверу
MT5WaitForTerminal()

# установим таймзону в UTC
timezone = pytz.timezone("Etc/UTC")
# создадим объекты datetime в таймзоне UTC, чтобы не применялось смещение локальной таймзоны
utc_from = datetime(2019, 4, 5, tzinfo=timezone)
utc_to = datetime(2019, 4, 5, hour = 13, tzinfo=timezone)
# получим 10 баров с EURPLN M5 начиная с 01.04.2019 в таймзоне UTC
rates = MT5CopyRatesFrom("EURPLN", MT5_TIMEFRAME_M5, utc_from, utc_to)

# завершим подключение к терминалу MetaTrader 5
MT5Shutdown()
# выведем каждый элемент полученных данных на новой строке
print("Выведем полученные данные как есть")
for rate in rates:
    print(rate)

# создадим из полученных данных DataFrame
rates_frame = pd.DataFrame(list(rates),
                           columns=['time', 'open', 'low', 'high', 'close', 'tick_volume'])
# выведем данные
print("\nВыведем датафрейм с данными")
print(rates_frame) # видим, что Python представляет время открытия баров в локальной таймзоне

# получим для локального компьютера смещение от времени UTC
UTC_OFFSET_TIMDELTA = datetime.utcnow() - datetime.now()

# создадим простую функцию, которая влоб корректирует смещение
def local_to_utc(dt):
    return dt + UTC_OFFSET_TIMDELTA

# применим смещение для столбца time в датафрейме rates_frame
rates_frame['time'] = rates_frame.apply(lambda rate: local_to_utc(rate['time']), axis=1)

# еще раз выведем данные и убедимся, что теперь время открытия изменилось
print("\nВыведем датафрейм после корректировки времени")
print(rates_frame)

```

Результат:

Выведем полученные данные как есть

```

MT5Rate(time=datetime.datetime(2019, 4, 1, 3, 30), open=4.2988, low=4.2993, high=4.2988)
MT5Rate(time=datetime.datetime(2019, 4, 1, 3, 35), open=4.2989, low=4.2989, high=4.2988)
MT5Rate(time=datetime.datetime(2019, 4, 1, 3, 40), open=4.2989, low=4.2991, high=4.2988)
MT5Rate(time=datetime.datetime(2019, 4, 1, 3, 45), open=4.2989, low=4.2989, high=4.2988)
MT5Rate(time=datetime.datetime(2019, 4, 1, 3, 50), open=4.2989, low=4.2989, high=4.2988)
MT5Rate(time=datetime.datetime(2019, 4, 1, 3, 55), open=4.2989, low=4.2989, high=4.2988)
MT5Rate(time=datetime.datetime(2019, 4, 1, 4, 0), open=4.2989, low=4.2995, high=4.2988)
MT5Rate(time=datetime.datetime(2019, 4, 1, 4, 5), open=4.2995, low=4.2998, high=4.2998)
MT5Rate(time=datetime.datetime(2019, 4, 1, 4, 10), open=4.2998, low=4.2998, high=4.2998)
MT5Rate(time=datetime.datetime(2019, 4, 1, 4, 15), open=4.2992, low=4.2995, high=4.2998)

```

Выведем из датафрейма первые 10 значений

	time	open	low	high	close	tick_volume	spread	real_volu
0	2019-04-01 03:30:00	4.2988	4.2993	4.29860	4.2989	853	540	
1	2019-04-01 03:35:00	4.2989	4.2989	4.29890	4.2989	1	630	
2	2019-04-01 03:40:00	4.2989	4.2991	4.29870	4.2989	10	560	
3	2019-04-01 03:45:00	4.2989	4.2989	4.29890	4.2989	2	600	
4	2019-04-01 03:50:00	4.2989	4.2989	4.29890	4.2989	4	640	
5	2019-04-01 03:55:00	4.2989	4.2989	4.29890	4.2989	4	640	
6	2019-04-01 04:00:00	4.2989	4.2995	4.29872	4.2994	40	500	
7	2019-04-01 04:05:00	4.2995	4.2998	4.29900	4.2998	38	440	
8	2019-04-01 04:10:00	4.2998	4.2998	4.29910	4.2992	35	440	
9	2019-04-01 04:15:00	4.2992	4.2995	4.29890	4.2993	379	500	

Выведем из датафрейма первые 10 значений после корректировки времени

	time	open	low	high	close	tick_volume	spread	real_volu
0	2019-04-01 00:30:00	4.2988	4.2993	4.29860	4.2989	853	540	
1	2019-04-01 00:35:00	4.2989	4.2989	4.29890	4.2989	1	630	
2	2019-04-01 00:40:00	4.2989	4.2991	4.29870	4.2989	10	560	
3	2019-04-01 00:45:00	4.2989	4.2989	4.29890	4.2989	2	600	
4	2019-04-01 00:50:00	4.2989	4.2989	4.29890	4.2989	4	640	
5	2019-04-01 00:55:00	4.2989	4.2989	4.29890	4.2989	4	640	
6	2019-04-01 01:00:00	4.2989	4.2995	4.29872	4.2994	40	500	
7	2019-04-01 01:05:00	4.2995	4.2998	4.29900	4.2998	38	440	
8	2019-04-01 01:10:00	4.2998	4.2998	4.29910	4.2992	35	440	
9	2019-04-01 01:15:00	4.2992	4.2995	4.29890	4.2993	379	500	

#### Смотри также

[CopyRates](#), [MT5CopyRatesFrom](#), [MT5CopyRatesRange](#), [MT5CopyTicksFrom](#), [MT5CopyTicksRange](#)

## MT5CopyTicksFrom

Получает тики из терминала MetaTrader 5, начиная с указанной даты.

```
MT5CopyTicksFrom(
    symbol,           // имя символа
    from,             // дата, с которой запрашиваются тики
    count,            // количество запрашиваемы тиков
    flags             // комбинация флагов, определяющая тип запрашиваемых тиков
)
```

### Параметры

*symbol*

[in] Имя финансового инструмента, например, "EURUSD".

*from*

[in] Дата, начиная с которой запрашиваются тики. Задается объектом `datetime` или в виде количества секунд, прошедших с 1970.01.01.

*count*

[in] Количество тиков, которое необходимо получить.

*flags*

[in] Флаг, определяющий тип запрашиваемых тиков. [MT5\\_COPY\\_TICKS\\_INFO](#) - тики, вызванные изменениями Bid и/или Ask, [MT5\\_COPY\\_TICKS\\_TRADE](#) - тики с изменения Last и Volume, [MT5\\_COPY\\_TICKS\\_ALL](#) - все тики. Значения флагов описаны в перечислении [MT5\\_COPY\\_TICKS](#).

### Возвращаемое значение

Возвращает тики в виде кортежей (time, bid, ask, last, flags). В кортеже переменная *flags* может быть комбинацией флагов из перечисления [MT5\\_TICK\\_FLAG](#).

### Примечание

Для дополнительной информации смотрите функцию [CopyTicks](#).

Python при создании объекта `datetime` использует таймзону локального времени, в то время как терминал MetaTrader 5 хранит время тиков и открытия баров в UTC таймзоне (без смещения). Поэтому, для выполнения функций, использующих время, необходимо создавать `datetime` в UTC-времени. Данные, полученные из терминала MetaTrader 5, имеют UTC-время, но при попытке вывести их на печать, Python опять применит смещение для локального времени. Поэтому, полученные данные также необходимо корректировать для визуального представления.

[MT5\\_COPY\\_TICKS](#) определяет типы тиков, которые могут быть запрошены с помощью функций [MT5CopyTicksFrom\(\)](#) и [MT5CopyTicksRange\(\)](#).

Идентификатор	Описание
<a href="#">MT5_COPY_TICKS_ALL</a>	все тики

MT5_COPY_TICKS_INFO	тики, содержащие изменения цен Bid и/или Ask
MT5_COPY_TICKS_TRADE	тики, содержащие изменения цены Last и/или объема (Volume)

**MT5\_TICK\_FLAG** определяет возможные флаги для тиков. Данные флаги используются для описания тиков, полученных функциями [MT5CopyTicksFrom\(\)](#) и [MT5CopyTicksRange\(\)](#).

Идентификатор	Описание
MT5_TICK_FLAG_BID	изменилась цена Bid
MT5_TICK_FLAG_ASK	изменялась цена Ask
MT5_TICK_FLAG_LAST	изменилась цена Last
MT5_TICK_FLAG_VOLUME	изменился объем (Volume)
MT5_TICK_FLAG_BUY	изменилась цена последней покупки (Buy)
MT5_TICK_FLAG_SELL	изменилась цена последней продажи (Sell)

### Пример:

```
from datetime import datetime
from MetaTrader5 import *
# импортируем модуль pandas для вывода полученных данных в табличной форме
import pandas as pd
pd.set_option('display.max_columns', 500) # сколько столбцов показываем
pd.set_option('display.width', 1500)       # макс. ширина таблицы для показа
# импортируем модуль pytz для работы с таймзоной
import pytz

# установим подключение к терминалу MetaTrader 5
MT5Initialize()
# подождем пока MetaTrader 5 подключится к торговому серверу
MT5WaitForTerminal()

# установим таймзону в UTC
timezone = pytz.timezone("Etc/UTC")
# создадим объект datetime в таймзоне UTC, чтобы не применялось смещение локальной таймзоны
utc_from = datetime(2019, 4, 1, tzinfo=timezone)
# запросим 100000 тиков по EURUSD с 01.04.2019 в таймзоне UTC
ticks = MT5CopyTicksFrom("EURUSD", utc_from, 100000, MT5_COPY_TICKS_ALL)
print("Получено тиков:", len(ticks))

# завершим подключение к терминалу MetaTrader 5
MT5Shutdown()
# выведем данные каждого тика на новой строке
print("Выведем полученные тики как есть")
```

```

count = 0
for tick in ticks:
    print(tick)
    if(count >= 10):
        break

# создадим из полученных данных DataFrame
ticks_frame = pd.DataFrame(list(ticks),
                            columns=['time', 'bid', 'ask', 'last', 'volume', 'flags'])

# выведем данные
print("\nВыведем датафрейм с тиками")
print(ticks_frame.head(10)) # видим, что Python представляет время тиков в локальной

# получим для локального компьютера смещение от времени UTC
UTC_OFFSET_TIMDELTA = datetime.utcnow() - datetime.now()

# создадим простую функцию, которая влоб корректирует смещение
def local_to_utc(dt):
    return dt + UTC_OFFSET_TIMDELTA

# применим смещение для столбца time в датафрейме ticks_frame
ticks_frame['time'] = ticks_frame.apply(lambda tick: local_to_utc(tick['time']), axis=1)

# еще раз выведем данные и убедимся, что теперь время тиков изменилось
print("\nВыведем датафрейм с тиками после корректировки времени")
print(ticks_frame.head(10))

```

Результат:

Получено тиков: 100000

Выведем полученные тики как есть

```

MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 5, 745000), bid=1.12258, ask=1.12339,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 26, 155000), bid=1.12260, ask=1.12339,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 26, 497000), bid=1.12272, ask=1.12339,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 34, 327000), bid=1.12268, ask=1.12336,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 35, 61000), bid=1.12268, ask=1.12339,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 35, 369000), bid=1.12258, ask=1.12339,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 40, 577000), bid=1.12258, ask=1.12336,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 46, 494000), bid=1.1225, ask=1.12336,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 47, 288000), bid=1.12248, ask=1.12336,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 56, 114000), bid=1.1225, ask=1.12336,

```

Выведем датафрейм с тиками

	time	bid	ask	last	volume	flags
0	2019-04-01 03:02:05.745	1.12258	1.12339	0.0	0.0	134
1	2019-04-01 03:02:26.155	1.12261	1.12339	0.0	0.0	130
2	2019-04-01 03:02:26.497	1.12272	1.12339	0.0	0.0	130
3	2019-04-01 03:02:34.327	1.12268	1.12336	0.0	0.0	134
4	2019-04-01 03:02:35.061	1.12268	1.12339	0.0	0.0	4
5	2019-04-01 03:02:35.369	1.12258	1.12339	0.0	0.0	130
6	2019-04-01 03:02:40.577	1.12258	1.12336	0.0	0.0	4
7	2019-04-01 03:02:46.494	1.12250	1.12336	0.0	0.0	130
8	2019-04-01 03:02:47.288	1.12248	1.12336	0.0	0.0	130
9	2019-04-01 03:02:56.114	1.12250	1.12336	0.0	0.0	130

Выведем датафрейм с тиками после корректировки времени

	time	bid	ask	last	volume	flags
0	2019-04-01 00:02:05.745	1.12258	1.12339	0.0	0.0	134
1	2019-04-01 00:02:26.155	1.12261	1.12339	0.0	0.0	130
2	2019-04-01 00:02:26.497	1.12272	1.12339	0.0	0.0	130
3	2019-04-01 00:02:34.327	1.12268	1.12336	0.0	0.0	134
4	2019-04-01 00:02:35.061	1.12268	1.12339	0.0	0.0	4
5	2019-04-01 00:02:35.369	1.12258	1.12339	0.0	0.0	130
6	2019-04-01 00:02:40.577	1.12258	1.12336	0.0	0.0	4
7	2019-04-01 00:02:46.494	1.12250	1.12336	0.0	0.0	130
8	2019-04-01 00:02:47.288	1.12248	1.12336	0.0	0.0	130
9	2019-04-01 00:02:56.114	1.12250	1.12336	0.0	0.0	130

#### Смотри также

[CopyRates](#), [MT5CopyRatesFromPos](#), [MT5CopyRatesRange](#), [MT5CopyTicksFrom](#), [MT5CopyTicksRange](#)

## MT5CopyTicksRange

Получает тики за указанный диапазон дат из терминала MetaTrader 5.

```
MT5CopyTicksRange (
    symbol,           // имя символа
    from,             // дата, с которой запрашиваются тики
    to,               // дата, по которую запрашиваются тики
    flags            // комбинация флагов, определяющая тип запрашиваемых тиков
)
```

### Параметры

*symbol*

[in] Имя финансового инструмента, например, "EURUSD".

*from*

[in] Дата, начиная с которой запрашиваются тики. Задается объектом datetime или в виде количества секунд, прошедших с 1970.01.01.

*to*

[in] Дата, по которую запрашиваются тики. Задается объектом datetime или в виде количества секунд, прошедших с 1970.01.01.

*flags*

[in] Флаг, определяющий тип запрашиваемых тиков. [MT5\\_COPY\\_TICKS\\_INFO](#) - тики, вызванные изменениями Bid и/или Ask, [MT5\\_COPY\\_TICKS\\_TRADE](#) - тики с изменениями Last и Volume, [MT5\\_COPY\\_TICKS\\_ALL](#) - все тики. Значения флагов описаны в перечислении [MT5\\_COPY\\_TICKS](#).

### Возвращаемое значение

Возвращает тики в виде кортежей (time, bid, ask, last, flags). В кортеже переменная flags может быть комбинацией флагов из перечисления [MT5\\_TICK\\_FLAG](#).

### Примечание

Для дополнительной информации смотрите функцию [CopyTicks](#).

Python при создании объекта datetime использует таймзону локального времени, в то время как терминал MetaTrader 5 хранит время тиков и открытия баров в UTC таймзоне (без смещения). Поэтому, для выполнения функций, использующих время, необходимо создавать datetime в UTC-времени. Данные, полученные из терминала MetaTrader 5, имеют UTC-время, но при попытке вывести их на печать, Python опять применит смещение для локального времени. Поэтому, полученные данные также необходимо корректировать для визуального представления.

### Пример:

```
from datetime import datetime
from MetaTrader5 import *
# импортируем модуль pandas для вывода полученных данных в табличной форме
import pandas as pd
pd.set_option('display.max_columns', 500) # сколько столбцов показываем
```

```

pd.set_option('display.width', 1500)      # макс. ширина таблицы для показа
# импортируем модуль pytz для работы с таймзоной
import pytz

# установим подключение к терминалу MetaTrader 5
MT5Initialize()
# подождем пока MetaTrader 5 подключится к торговому серверу
MT5WaitForTerminal()

# установим таймзону в UTC
timezone = pytz.timezone("Etc/UTC")
# создадим объекты datetime в таймзоне UTC, чтобы не применялось смещение локальной таймзоны
utc_from = datetime(2019, 4, 1, tzinfo=timezone)
utc_to = datetime(2019, 4, 5, tzinfo=timezone)
# запросим тики по AUDUSD в интервале 01.04.2019 - 05.04.2019
ticks = MT5CopyTicksFrom("EURUSD", utc_from, utc_to, MT5_COPY_TICKS_ALL)
print("Получено тиков:", len(ticks))

# завершим подключение к терминалу MetaTrader 5
MT5Shutdown()
# выведем данные каждого тика на новой строке
print("Выведем полученные тики как есть")
count = 0
for tick in ticks:
    print(tick)
    if(count >= 10):
        break

# создадим из полученных данных DataFrame
ticks_frame = pd.DataFrame(list(ticks),
                           columns=['time', 'bid', 'ask', 'last', 'volume', 'flags'])
# выведем данные
print("\nВыведем датафрейм с тиками")
print(ticks_frame.head(10)) # видим, что Python представляет время тиков в локальной таймзоне

# получим для локального компьютера смещение от времени UTC
UTC_OFFSET_TIMDELTA = datetime.utcnow() - datetime.now()

# создадим простую функцию, которая влоб корректирует смещение
def local_to_utc(dt):
    return dt + UTC_OFFSET_TIMDELTA

# применим смещение для столбца time в датафрейме ticks_frame
ticks_frame['time'] = ticks_frame.apply(lambda tick: local_to_utc(tick['time']), axis=1)

# еще раз выведем данные и убедимся, что теперь время тиков изменилось
print("\nВыведем датафрейм с тиками после корректировки времени")
print(ticks_frame.head(10))

```

**Результат:**

Получено тиков: 243374

Выведем полученные тики как есть

```

MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 2, 759000), bid=0.71155, ask=0.71223,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 7, 824000), bid=0.71154, ask=0.71222,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 27, 818000), bid=0.71155, ask=0.71221,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 43, 794000), bid=0.71159, ask=0.71204,
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 44, 67000), bid=0.71145, ask=0.71209,

```

```
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 44, 140000), bid=0.71157, ask=0.71221)
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 44, 432000), bid=0.71157, ask=0.71225)
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 44, 872000), bid=0.71159, ask=0.71204)
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 45, 164000), bid=0.71148, ask=0.71211)
MT5Tick(time=datetime.datetime(2019, 4, 1, 3, 2, 45, 244000), bid=0.71158, ask=0.71221)
```

Выведем датафрейм с тиками

	time	bid	ask	last	volume	flags
0	2019-04-01 03:02:02.759	0.71155	0.71223	0.0	0.0	134
1	2019-04-01 03:02:07.824	0.71154	0.71222	0.0	0.0	134
2	2019-04-01 03:02:27.818	0.71155	0.71223	0.0	0.0	134
3	2019-04-01 03:02:43.794	0.71159	0.71204	0.0	0.0	134
4	2019-04-01 03:02:44.067	0.71145	0.71209	0.0	0.0	134
5	2019-04-01 03:02:44.140	0.71157	0.71223	0.0	0.0	134
6	2019-04-01 03:02:44.432	0.71157	0.71225	0.0	0.0	4
7	2019-04-01 03:02:44.872	0.71159	0.71204	0.0	0.0	134
8	2019-04-01 03:02:45.164	0.71148	0.71210	0.0	0.0	134
9	2019-04-01 03:02:45.244	0.71158	0.71223	0.0	0.0	134

Выведем датафрейм с тиками после корректировки времени

	time	bid	ask	last	volume	flags
0	2019-04-01 00:02:02.759	0.71155	0.71223	0.0	0.0	134
1	2019-04-01 00:02:07.824	0.71154	0.71222	0.0	0.0	134
2	2019-04-01 00:02:27.818	0.71155	0.71223	0.0	0.0	134
3	2019-04-01 00:02:43.794	0.71159	0.71204	0.0	0.0	134
4	2019-04-01 00:02:44.067	0.71145	0.71209	0.0	0.0	134
5	2019-04-01 00:02:44.140	0.71157	0.71223	0.0	0.0	134
6	2019-04-01 00:02:44.432	0.71157	0.71225	0.0	0.0	4
7	2019-04-01 00:02:44.872	0.71159	0.71204	0.0	0.0	134
8	2019-04-01 00:02:45.164	0.71148	0.71210	0.0	0.0	134
9	2019-04-01 00:02:45.244	0.71158	0.71223	0.0	0.0	134

## Смотри также

[CopyRates](#), [MT5CopyRatesFromPos](#), [MT5CopyRatesRange](#), [MT5CopyTicksFrom](#), [MT5CopyTicksRange](#)

## Стандартная библиотека

Эта группа разделов содержит технические детали работы со стандартной библиотекой MQL5 и описания всех ее ключевых компонентов.

Стандартная библиотека MQL5 написана на языке MQL5 и предназначена для облегчения написания программ (индикаторов, скриптов, экспертов) конечным пользователям. Библиотека обеспечивает удобный доступ к большинству внутренних функций MQL5.

Стандартная библиотека MQL5 размещается в рабочем каталоге терминала в папке `Include`.

Раздел	Размещение
<a href="#">Математика</a>	<code>Include\Math\</code>
<a href="#">OpenCL</a>	<code>Include\OpenCL\</code>
<a href="#">Базовый класс CObject</a>	<code>Include\</code>
<a href="#">Коллекции данных</a>	<code>Include\Arrays\</code>
<a href="#">Шаблонные коллекции данных</a>	<code>Include\Generic\</code>
<a href="#">Файлы</a>	<code>Include\Files\</code>
<a href="#">Строки</a>	<code>Include\Strings\</code>
<a href="#">Графические объекты</a>	<code>Include\Objects\</code>
<a href="#">Пользовательская графика</a>	<code>Include\Canvas\</code>
<a href="#">Ценовые графики</a>	<code>Include\Charts\</code>
<a href="#">Научные графики</a>	<code>Include\Graphics\</code>
<a href="#">Индикаторы</a>	<code>Include\Indicators\</code>
<a href="#">Торговые классы</a>	<code>Include\Trade\</code>
<a href="#">Модули стратегий</a>	<code>Include\Expert\</code>
<a href="#">Панели и диалоги</a>	<code>Include\Controls\</code>

Для проведения вычислений в разных областях математики предлагается несколько библиотек:

- [Статистика](#) - функции для работы с различными распределениями из теории вероятности
- [Нечеткая логика](#) - библиотека нечеткой логики, в которой реализованы системы нечеткого логического вывода Мамдани и Сугено
- [ALGLIB](#) - анализ данных (кластеризация, деревья решений, линейная регрессия, нейронные сети), решения дифференциальных уравнений, преобразования Фурье, численное интегрирование, задачи оптимизации, статистический анализ и многое другое.

## Статистика

Статистическая библиотека предназначена для удобной работы с основными статистическими распределениями.

Для каждого распределения в библиотеке представлено 5 функций:

1. Расчет плотности распределения - функции вида `MathProbabilityDensityX()`
2. Расчет вероятностей - функции вида `MathCumulativeDistributionX()`
3. Расчет квантилей распределений - функции вида `MathQuantileX()`
4. Генерация случайных чисел с заданным распределением - функции вида `MathRandomX()`
5. Расчет теоретических моментов распределений - функции вида `MathMomentsX()`

Кроме расчета значений для отдельных случайных величин в библиотеке представлены также перегрузки функций, которые производят такие же расчеты на массивах.

- [Статистические характеристики](#)
- [Нормальное распределение](#)
- [Логнормальное распределение](#)
- [Бета-распределение](#)
- [Нецентральное бета-распределение](#)
- [Гамма-распределение](#)
- [Распределение хи-квадрат](#)
- [Нецентральное распределение хи-квадрат](#)
- [Экспоненциальное распределение](#)
- [F-распределение](#)
- [Нецентральное F-распределение](#)
- [T-распределение](#)
- [Нецентральное T-распределение](#)
- [Логистическое распределение](#)
- [Распределение Коши](#)
- [Равномерное распределение](#)
- [Распределение Вейбулла](#)
- [Биномиальное распределение](#)
- [Отрицательное биномиальное распределение](#)
- [Геометрическое распределение](#)
- [Гипергеометрическое распределение](#)
- [Распределение Пуассона](#)
- [Вспомогательные функции](#)

Пример:

```

//+-----+
//|                               NormalDistributionExample.mq5 |
//| Copyright 2016, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- подключаем функции для расчета нормального распределения
#include <Math\Stat\Normal.mqh>
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- задаем параметры нормального распределения
    double mu=5.0;
    double sigma=1.0;
    PrintFormat("Нормальное распределение с параметрами mu=%G и sigma=%G, примеры вычислений");
//--- задаем интервал
    double x1=mu-sigma;
    double x2=mu+sigma;
//--- переменные для расчета вероятности
    double cdf1,cdf2,probability;
//--- переменные для кода ошибки
    int error_code1,error_code2;
//--- рассчитаем значения функции распределения
    cdf1=MathCumulativeDistributionNormal(x1,mu,sigma,error_code1);
    cdf2=MathCumulativeDistributionNormal(x2,mu,sigma,error_code2);
//--- проверим код ошибок
    if(error_code1==ERR_OK && error_code2==ERR_OK)
    {
//--- рассчитаем вероятность случайной величины в диапазоне
        probability=cdf2-cdf1;
//--- выводим результат
        PrintFormat("1. Вычислить в диапазоне %.5f<x<%.5f вероятность случайной величины");
        PrintFormat(" Ответ: Probability = %5.8f",probability);
    }

//--- Найдем интервал значений случайной величины x, соответствующий 95% доверительной вероятности
    probability=0.95; // задаем доверительную вероятность
//--- задаем вероятности на границах интервала
    double p1=(1.0-probability)*0.5;
    double p2=probability+(1.0-probability)*0.5;
//--- вычислим границы интервала
    x1=MathQuantileNormal(p1,mu,sigma,error_code1);
    x2=MathQuantileNormal(p2,mu,sigma,error_code2);
//--- проверим код ошибок
    if(error_code1==ERR_OK && error_code2==ERR_OK)
    {
//--- выводим результат
        PrintFormat("2. Для доверительного интервала = %.2f найти диапазон случайной величины");
        PrintFormat(" Ответ: диапазон %5.8f <= x <=%5.8f",x1,x2);
    }

    PrintFormat("3. Вычислить рассчитанные и теоретические 4 первые момента распределения");
//--- Сгенерируем массив случайных чисел, рассчитаем первые 4 момента и сравним с теоретическими
    int data_count=1000000; // задаем количество значений и подготавливаем массив
    double data[];
    ArrayResize(data,data_count);
//--- генерируем случайные значения и сохраняем их в массив
}

```

```
for(int i=0; i<data_count; i++)
{
    data[i]=MathRandomNormal(mu,sigma,error_code1);
}
//--- задаем индекс начального значения и количество данных для расчета
int start=0;
int count=data_count;
//--- вычислим первые 4 момента сгенерированных значений
double mean=MathMean(data,start,count);
double variance=MathVariance(data,start,count);
double skewness=MathSkewness(data,start,count);
double kurtosis=MathKurtosis(data,start,count);
//--- переменные для теоретических моментов
double normal_mean=0;
double normal_variance=0;
double normal_skewness=0;
double normal_kurtosis=0;
//--- выводим значения рассчитанных моментов
PrintFormat("          Mean          Variance          Skewness          Kurtos");
PrintFormat("Calculated  %.10f  %.10f  %.10f  %.10f",mean,variance,skewness,kurtosis);
//--- рассчитываем теоретические значения моментов и сравниваем с полученными
if(MathMomentsNormal(mu,sigma,normal_mean,normal_variance,normal_skewness,normal_kurtosis))
{
    PrintFormat("Theoretical  %.10f  %.10f  %.10f  %.10f",normal_mean,normal_variance,normal_skewness,normal_kurtosis);
    PrintFormat("Difference   %.10f  %.10f  %.10f  %.10f",mean-normal_mean,mean-normal_variance,mean-normal_skewness,mean-normal_kurtosis);
}
}
```

## Статистические характеристики

Эта группа функций рассчитывает статистические характеристики элементов массива:

- среднее,
- дисперсию,
- коэффициент асимметрии,
- коэффициент эксцесса,
- медиану,
- среднеквадратичное и
- стандартные отклонения.

Функция	Описание
<a href="#">MathMean</a>	Рассчитывает среднее значение (первый момент) элементов массива
<a href="#">MathVariance</a>	Рассчитывает дисперсию (второй момент) элементов массива
<a href="#">MathSkewness</a>	Рассчитывает коэффициент асимметрии (третий момент) элементов массива
<a href="#">MathKurtosis</a>	Рассчитывает коэффициент эксцесса (четвертый момент) элементов массива
<a href="#">MathMoments</a>	Рассчитывает первые 4 момента (среднее, дисперсия, коэффициент асимметрии, коэффициент эксцесса) элементов массива
<a href="#">MathMedian</a>	Рассчитывает медианное значение элементов массива
<a href="#">MathStandardDeviation</a>	Рассчитывает стандартное отклонение элементов массива
<a href="#">MathAverageDeviation</a>	Рассчитывает среднее отклонение элементов массива

## MathMean

Рассчитывает среднее значение (первый момент) элементов массива. Аналог [mean\(\)](#) в R.

```
double MathMean(
    const double& array[] // массив с данными
);
```

### Параметры

*array*

[in] Массив с данными для расчета среднего значения.

*start=0*

[in] Начальный индекс для расчета.

*count=WHOLE\_ARRAY*

[in] Количество элементов для расчета.

### Возвращаемое значение

Среднее значение элементов массива. В случае ошибки возвращает [NaN](#) (не число).

## MathVariance

Рассчитывает дисперсию (второй момент) элементов массива. Аналог [var\(\)](#) в R.

```
double MathVariance(
    const double& array[] // массив с данными
);
```

### Параметры

*array*

[in] Массив с данными для расчета.

*start=0*

[in] Начальный индекс для расчета.

*count=WHOLE\_ARRAY*

[in] Количество элементов для расчета.

### Возвращаемое значение

Дисперсия элементов массива. В случае ошибки возвращает [NaN](#) (не число).

## MathSkewness

Рассчитывает коэффициент асимметрии (третий момент) элементов массива. Аналог [skewness\(\)](#) в R (библиотека e1071).

```
double MathSkewness(
    const double& array[] // массив с данными
);
```

### Параметры

*array*  
[in] Массив с данными для расчета.

*start=0*  
[in] Начальный индекс для расчета.

*count=WHOLE\_ARRAY*  
[in] Количество элементов для расчета.

### Возвращаемое значение

Коэффициент асимметрии элементов массива. В случае ошибки возвращает [NaN](#) (не число).

## MathKurtosis

Рассчитывает коэффициент эксцесса (четвертый момент) элементов массива. Аналог [kurtosis\(\)](#) в R (библиотека e1071).

```
double MathKurtosis(
    const double& array[] // массив с данными
);
```

### Параметры

*array*  
[in] Массив с данными для расчета.

*start=0*  
[in] Начальный индекс для расчета.

*count=WHOLE\_ARRAY*  
[in] Количество элементов для расчета.

### Возвращаемое значение

Коэффициент эксцесса элементов массива. В случае ошибки возвращает [NaN](#) (не число).

### Примечание

Расчет коэффициента эксцесса производится относительно нормального распределения (excess kurtosis=kurtosis-3), т.е. коэффициент эксцесса нормального распределения равен нулю.

Он положителен, если пик распределения около математического ожидания острый, и отрицателен, если вершина гладкая.

## MathMoments

Рассчитывает первые 4 момента (среднее, дисперсия, коэффициент асимметрии, коэффициент эксцесса) элементов массива.

```
double MathMoments(
    const double& array[],           // массив с данными
    double& mean,                   // среднее значение (1-ый момент)
    double& variance,              // дисперсия (2-ой момент)
    double& skewness,               // коэффициент асимметрии (3-ий момент)
    double& kurtosis,                // коэффициент эксцесса (4-ый момент)
    const int start=0,               // начальный индекс
    const int count=WHOLE_ARRAY     // количество элементов
);
```

### Параметры

*array*

[in] Массив с данными для расчета.

*mean*

[out] Переменная для среднего значения (1 момент)

*variance*

[out] Переменная для дисперсии (2 момент)

*skewness*

[out] Переменная для коэффициента асимметрии (3 момент)

*kurtosis*

[out] Переменная для коэффициента эксцесса (4 момент)

*start=0*

[in] Начальный индекс для расчета.

*count=WHOLE\_ARRAY*

[in] Количество элементов для расчета.

### Возвращаемое значение

Возвращает true, если моменты успешно рассчитаны, иначе false.

### Примечание

Расчет коэффициента эксцесса производится относительно нормального распределения (excess kurtosis=kurtosis-3), т.е. коэффициент эксцесса нормального распределения равен нулю.

Он положителен, если пик распределения около математического ожидания острый, и отрицателен, если вершина гладкая.

## MathMedian

Рассчитывает медианное значение элементов массива. Аналог [median\(\)](#) в R.

```
double MathMedian(
    const double& array[] // массив с данными
);
```

### Параметры

*array*

[in] Массив с данными для расчета.

*start=0*

[in] Начальный индекс для расчета.

*count=WHOLE\_ARRAY*

[in] Количество элементов для расчета.

### Возвращаемое значение

Медианное значение элементов массива. В случае ошибки возвращает [NaN](#) (не число).

## MathStandardDeviation

Рассчитывает стандартное отклонение элементов массива. Аналог [sd\(\)](#) в R.

```
double MathStandardDeviation(
    const double& array[] // массив с данными
);
```

### Параметры

*array*

[in] Массив с данными для расчета.

*start=0*

[in] Начальный индекс для расчета.

*count=WHOLE\_ARRAY*

[in] Количество элементов для расчета.

### Возвращаемое значение

Стандартное отклонение элементов массива. В случае ошибки возвращает [NaN](#) (не число).

## MathAverageDeviation

Рассчитывает среднее отклонение элементов массива. Аналог [aad\(\)](#) в R.

```
double MathAverageDeviation(
    const double& array[] // массив с данными
);
```

### Параметры

*array*

[in] Массив с данными для расчета.

*start=0*

[in] Начальный индекс для расчета.

*count=WHOLE\_ARRAY*

[in] Количество элементов для расчета.

### Возвращаемое значение

Среднее отклонение элементов массива. В случае ошибки возвращает [NaN](#) (не число).

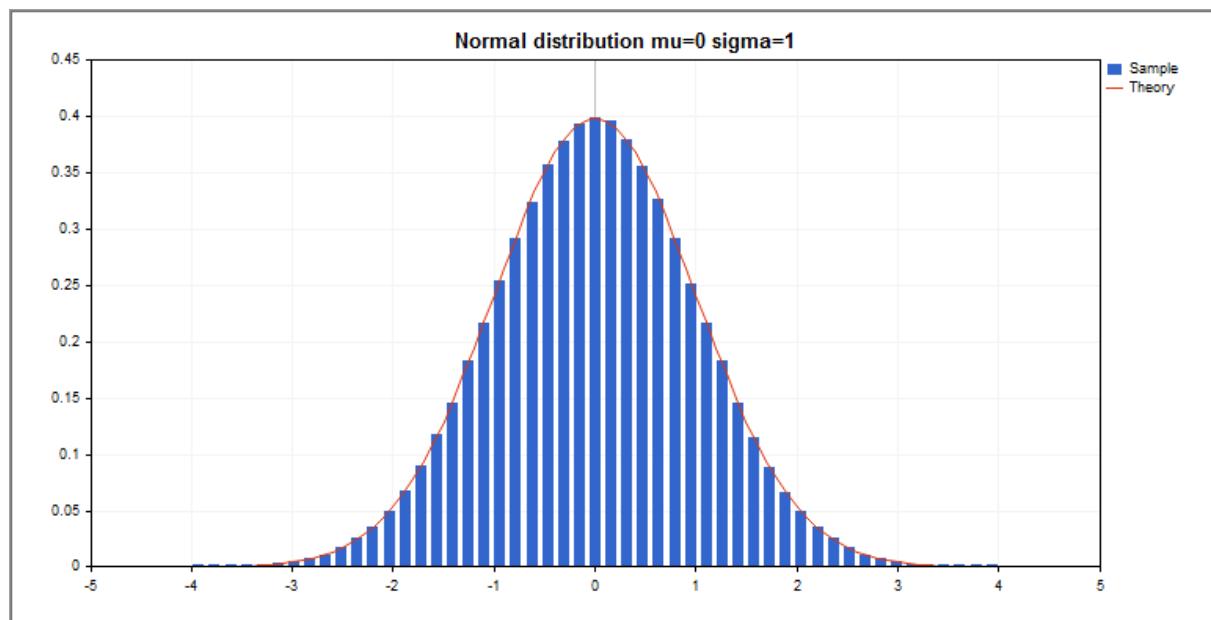
## Нормальное распределение

В данном разделе представлены функции для работы с нормальным распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по нормальному закону. Распределение описывается следующей формулой:

$$f_{Normal}(x | \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

где:

- $x$  — значение случайной величины
- $\mu$  — математическое ожидание
- $\sigma$  — среднеквадратическое отклонение



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityNormal</a>	Рассчитывает плотность вероятности нормального распределения
<a href="#">MathCumulativeDistributionNormal</a>	Рассчитывает значение функции нормального распределения вероятностей
<a href="#">MathQuantileNormal</a>	Рассчитывает значение обратной функции нормального распределения для заданной вероятности
<a href="#">MathRandomNormal</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по нормальному закону

MathMomentsNormal

Рассчитывает теоретические численные значения первых 4 моментов нормального распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Normal.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mean_value=0; // математическое ожидание (mean)
input double std_dev=1; // среднеквадратическое отклонение (standard deviation)
//+-----+
//| Script program start function |+
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
long chart=0;
string name="GraphicNormal";
int n=1000000; // количество значений в выборке
int ncells=51; // количество интервалов в гистограмме
double x[]; // центры интервалов гистограммы
double y[]; // количество значений из выборки, попавших в интервал
double data[]; // выборка случайных значений
double max,min; // максимальное и минимальное значения в выборке
//--- получим выборку из нормального распределения
MathRandomNormal(mean_value,std_dev,n,data);
//--- рассчитаем данные для построения гистограммы
CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
double step;
GetMaxMinStepValues(max,min,step);
step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
double x2[];
double y2[];
MathSequence(min,max,step,x2);
MathProbabilityDensityNormal(x2,mean_value,std_dev,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
```

```

for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- ВЫВОДИМ ГРАФИКИ
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Normal distribution mu=%G sigma=%G",mean_value,sigma));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
//--- plot all curves
graphic.CurvePlotAll();
graphic.Update();
}

//+-----+
//| Calculate frequencies for data set | 
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```

```
//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityNormal

Рассчитывает плотность вероятности нормального распределения с параметрами *mu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNormal(
    const double x,           // значение случайной величины
    const double mu,          // параметр распределения mean (математическое ожидание)
    const double sigma,        // параметр распределения sigma (среднеквадратическое отклонение)
    const bool log_mode,      // расчет логарифма значения
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности нормального распределения с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает [false](#). Аналог [dnorm\(\)](#) в R.

```
double MathProbabilityDensityNormal(
    const double x[],         // массив со значениями случайной величины
    const double mu,          // параметр распределения mean (математическое ожидание)
    const double sigma,        // параметр распределения sigma (среднеквадратическое отклонение)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности нормального распределения с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает [false](#). Аналог [dnorm\(\)](#) в R.

```
bool MathProbabilityDensityNormal(
    const double& x[],        // массив со значениями случайной величины
    const double mu,           // параметр распределения mean (математическое ожидание)
    const double sigma,         // параметр распределения sigma (среднеквадратическое отклонение)
    const bool log_mode,       // расчет логарифма значения
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности нормального распределения с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает [false](#).

```
bool MathProbabilityDensityNormal(
    const double& x[],        // массив со значениями случайной величины
    const double mu,           // параметр распределения mean (математическое ожидание)
    const double sigma,         // параметр распределения sigma (среднеквадратическое отклонение)
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*mu*

[in] Параметр распределения mean (математическое ожидание).

*sigma*

[in] Параметр распределения sigma (среднеквадратическое отклонение).

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив для получения значений функции плотности вероятности.

## MathCumulativeDistributionNormal

Рассчитывает значение функции нормального распределения вероятностей с параметрами *mu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionNormal(
    const double x, // значение случайной величины
    const double mu, // математическое ожидание
    const double sigma, // среднеквадратическое отклонение
    const bool tail, // флаг расчета хвоста (tail)
    const bool log_mode, // расчет логарифма значения
    int& error_code // переменная для записи кода ошибки
);
```

Рассчитывает значение функции нормального распределения вероятностей с параметрами *mu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionNormal(
    const double x, // значение случайной величины
    const double mu, // математическое ожидание
    const double sigma, // среднеквадратическое отклонение
    int& error_code // переменная для записи кода ошибки
);
```

Рассчитывает значение функции нормального распределения вероятностей с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [dnorm\(\)](#) в R.

```
bool MathCumulativeDistributionNormal(
    const double& x[], // массив со значениями случайной величины
    const double mu, // математическое ожидание
    const double sigma, // среднеквадратическое отклонение
    const bool tail, // флаг расчета хвоста (tail)
    const bool log_mode, // расчет логарифма значения
    double& result[] // массив для значений функции вероятности
);
```

Рассчитывает значение функции нормального распределения вероятностей с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathCumulativeDistributionNormal(
    const double& x[], // массив со значениями случайной величины
    const double mu, // математическое ожидание
    const double sigma, // среднеквадратическое отклонение
    double& result[] // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*mu*

[in] Параметр распределения mean (математическое ожидание).

*sigma*

[in] Параметр распределения sigma (среднеквадратическое отклонение).

*tail*

[in] Флаг расчета. Если *tail*=true, то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode*=true, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив для получения значений функции вероятности.

## MathQuantileNormal

Рассчитывает для заданной вероятности *probability* значение обратной функции нормального распределения с параметрами *mu* и *sigma*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileNormal(
    const double probability,           // значение вероятности случайной величины
    const double mu,                   // математическое ожидание
    const double sigma,                // среднеквадратическое отклонение
    const bool tail,                  // флаг расчета хвоста (tail)
    const bool log_mode,              // расчет логарифма значения
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает для заданной вероятности *probability* значение обратной функции нормального распределения с параметрами *mu* и *sigma*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileNormal(
    const double probability,           // значение вероятности случайной величины
    const double mu,                   // математическое ожидание
    const double sigma,                // среднеквадратическое отклонение
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает для массива значений вероятности *probability[]* значения обратной функции нормального распределения с параметрами *mu* и *sigma*. В случае ошибки возвращает `false`. Аналог [qnorm\(\)](#) в R.

```
bool MathQuantileNormal(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double mu,                   // математическое ожидание
    const double sigma,                // среднеквадратическое отклонение
    const bool tail,                  // флаг расчета хвоста (tail)
    const bool log_mode,              // расчет логарифма значения
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает для массива значений вероятности *probability[]* значения обратной функции нормального распределения с параметрами *mu* и *sigma*. В случае ошибки возвращает `false`.

```
bool MathQuantileNormal(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double mu,                   // математическое ожидание
    const double sigma,                // среднеквадратическое отклонение
    double& result[]                 // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*mu*

[in] Параметр распределения mean (математическое ожидание).

*sigma*

[in] Параметр распределения sigma (среднеквадратическое отклонение).

*tail*

[in] Флаг расчета. Если `false`, то расчет ведется для вероятности 1.0 - *probability*.

*log\_mode*

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив для получения квантилей.

## MathRandomNormal

Генерирует псевдослучайную величину, распределенную по нормальному закону с параметрами *mu* и *sigma*. В случае ошибки возвращает [NaN](#).

```
double MathRandomNormal(
    const double mu,           // математическое ожидание
    const double sigma,         // среднеквадратическое отклонение
    int& error_code            // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по нормальному закону с параметрами *mu* и *sigma*. В случае ошибки возвращает `false`. Аналог [rnorm\(\)](#) в R.

```
bool MathRandomNormal(
    const double mu,           // математическое ожидание
    const double sigma,         // среднеквадратическое отклонение
    const int data_count,       // количество необходимых значений
    double& result[]           // массив для получения псевдослучайных величин
);
```

### Параметры

*mu*

[in] Параметр распределения mean (математическое ожидание).

*sigma*

[in] Параметр распределения sigma (среднеквадратическое отклонение).

*data\_count*

[in] Количество псевдослучайных значений, которые необходимо получить.

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsNormal

Рассчитывает теоретические численные значения первых 4 моментов нормального распределения.

```
double MathMomentsNormal(
    const double mu,           // математическое ожидание
    const double sigma,         // среднеквадратическое отклонение
    double& mean,              // переменная для среднего значения
    double& variance,          // переменная для дисперсии
    double& skewness,           // переменная для коэффициента асимметрии
    double& kurtosis,           // переменная для коэффициента эксцесса
    int& error_code            // переменная для записи кода ошибки
);
```

### Параметры

*mu*

[in] Параметр распределения mean (математическое ожидание).

*sigma*

[in] Параметр распределения sigma (среднеквадратическое отклонение).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если моменты успешно рассчитаны, иначе false.

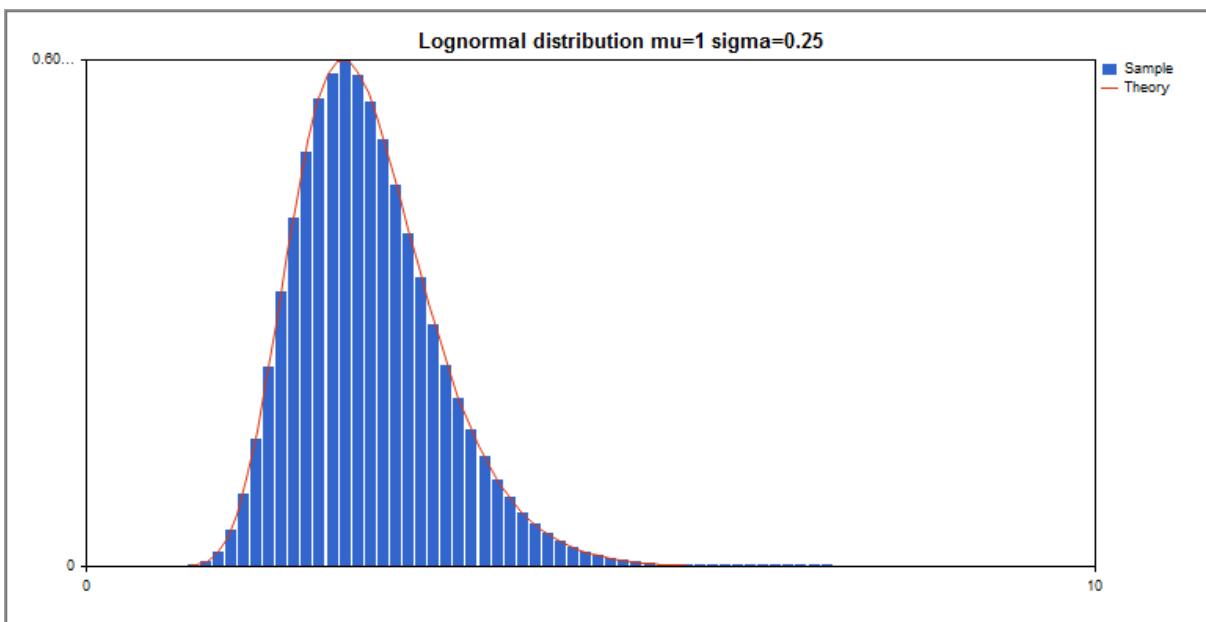
## Логнормальное распределение

В данном разделе представлены функции для работы с логнормальным распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по логнормальному закону. Логнормальное распределение описывается следующей формулой:

$$f_{Lognormal}(x | \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln(x)-\mu)^2}{2\sigma^2}}$$

где:

- $x$  – значение случайной величины
- $\mu$  – логарифм математического ожидания
- $\sigma$  – логарифм среднеквадратического отклонения



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityLognormal</a>	Рассчитывает плотность вероятности логнормального распределения
<a href="#">MathCumulativeDistributionLognormal</a>	Рассчитывает значение функции логнормального распределения вероятностей
<a href="#">MathQuantileLognormal</a>	Рассчитывает значение обратной функции логнормального распределения для заданной вероятности
<a href="#">MathRandomLognormal</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по логнормальному закону

[MathMomentsLognormal](#)

Рассчитывает теоретические численные значения первых 4 моментов логнормального распределения

Пример:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Lognormal.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mean_value=1.0; // логарифм математического ожидания (log mean)
input double std_dev=0.25; // логарифм среднеквадратического отклонения (log standard deviation)
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
long chart=0;
string name="GraphicNormal";
int n=1000000; // количество значений в выборке
int ncells=51; // количество интервалов в гистограмме
double x[]; // центры интервалов гистограммы
double y[]; // количество значений из выборки, попавших в интервал
double data[]; // выборка случайных значений
double max,min; // максимальное и минимальное значения в выборке
//--- получим выборку из логнормального распределения
MathRandomLognormal(mean_value,std_dev,n,data);
//--- рассчитаем данные для построения гистограммы
CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
double step;
GetMaxMinStepValues(max,min,step);
step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
double x2[];
double y2[];
MathSequence(min,max,step,x2);
MathProbabilityDensityLognormal(x2,mean_value,std_dev,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
```

```

        for(int i=0; i<ncells; i++)
            y[i]/=k;
//--- выводим графики
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Lognormal distribution mu=%G sigma=%G",mean_val,sigma));
graphic.BackgroundMainSize(16);
//--- отключим автомасштабирование оси Y
graphic.YAxis().AutoScale(false);
graphic.YAxis().Max(theor_max);
graphic.YAxis().Min(0);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
    }
}

```

```
        frequency[ind]++;
    }
    return (true);
}
//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityLognormal

Рассчитывает плотность вероятности логнормального распределения с параметрами *mu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityLognormal(
    const double x,                      // значение случайной величины
    const double mu,                     // логарифм математического ожидания (log mean)
    const double sigma,                  // логарифм среднеквадратического отклонения (log standard deviation)
    const bool log_mode,                // расчет логарифма значения, если log_mode=true, то
    int& error_code                   // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности логнормального распределения с параметрами *mu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityLognormal(
    const double x,                      // значение случайной величины
    const double mu,                     // логарифм математического ожидания (log mean)
    const double sigma,                  // логарифм среднеквадратического отклонения (log standard deviation)
    int& error_code                   // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности логнормального распределения с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает [NaN](#). Аналог [dlnorm\(\)](#) в R.

```
bool MathProbabilityDensityLognormal(
    const double& x[],                 // массив со значениями случайной величины
    const double mu,                   // логарифм математического ожидания (log mean)
    const double sigma,                // логарифм среднеквадратического отклонения (log standard deviation)
    const bool log_mode,              // расчет логарифма значения, если log_mode=true, то
    double& result[]                 // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности логнормального распределения с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityLognormal(
    const double& x[],                 // массив со значениями случайной величины
    const double mu,                   // логарифм математического ожидания (log mean)
    const double sigma,                // логарифм среднеквадратического отклонения (log standard deviation)
    double& result[]                 // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*mu*

[in] Логарифм математического ожидания (log\_mean).

*sigma*

[in] Логарифм среднеквадратического отклонения (log standard deviation).

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для получения значений функции плотности вероятности.

## MathCumulativeDistributionLognormal

Рассчитывает логнормальное распределение вероятностей с параметрами *mu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionLognormal(
    const double x,                                // значение случайной величины
    const double mu,                               // логарифм математического ожидания (log mean)
    const double sigma,                            // логарифм среднеквадратического отклонения (log standard deviation)
    const bool tail,                             // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,                         // расчет логарифма значения, если log_mode=true, то
    int& error_code                           // переменная для записи кода ошибки
);
```

Рассчитывает логнормальное распределение вероятностей с параметрами *mu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionLognormal(
    const double x,                                // значение случайной величины
    const double mu,                               // логарифм математического ожидания (log mean)
    const double sigma,                            // логарифм среднеквадратического отклонения (log standard deviation)
    int& error_code                           // переменная для записи кода ошибки
);
```

Рассчитывает логнормальное распределение вероятностей с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [plnorm\(\)](#) в R.

```
bool MathCumulativeDistributionLognormal(
    const double& x[],                           // массив со значениями случайной величины
    const double mu,                            // логарифм математического ожидания (log mean)
    const double sigma,                          // логарифм среднеквадратического отклонения (log standard deviation)
    const bool tail,                           // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,                      // расчет логарифма значения, если log_mode=true, то
    double& result[]                         // массив для значений функции вероятности
);
```

Рассчитывает логнормальное распределение вероятностей с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathCumulativeDistributionLognormal(
    const double& x[],                           // массив со значениями случайной величины
    const double mu,                            // логарифм математического ожидания (log mean)
    const double sigma,                          // логарифм среднеквадратического отклонения (log standard deviation)
    double& result[]                         // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*mu*

[in] Логарифм математического ожидания (log\_mean).

*sigma*

[in] Логарифм среднеквадратического отклонения (log standard deviation).

*tail*

[in] Флаг расчета, если true, то рассчитывается вероятность того, что случайная величина не превысит x.

*log\_mode*

[in] Флаг расчета логарифма значения. Если log\_mode=true, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для получения значений функции вероятности.

## MathQuantileLognormal

Рассчитывает значение обратной функции логнормального распределения с параметрами *mu* и *sigma* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileLognormal(
    const double probability,           // значение вероятности появления случайной величины
    const double mu,                   // логарифм математического ожидания (log mean)
    const double sigma,                // логарифм среднеквадратического отклонения (log standard deviation)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для вероятности log-probability
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции логнормального распределения с параметрами *mu* и *sigma* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileLognormal(
    const double probability,           // значение вероятности появления случайной величины
    const double mu,                   // логарифм математического ожидания (log mean)
    const double sigma,                // логарифм среднеквадратического отклонения (log standard deviation)
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции логнормального распределения с параметрами *mu* и *sigma* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*. Аналог [qlnorm\(\)](#) в R.

```
bool MathQuantileLognormal(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double mu,                  // логарифм математического ожидания (log mean)
    const double sigma,               // логарифм среднеквадратического отклонения (log standard deviation)
    const bool tail,                 // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,             // флаг расчета, если log_mode=true, то расчет ведется для вероятности log-probability
    double& result[]                // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции логнормального распределения с параметрами *mu* и *sigma* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*.

```
bool MathQuantileLognormal(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double mu,                  // логарифм математического ожидания (log mean)
    const double sigma,               // логарифм среднеквадратического отклонения (log standard deviation)
    double& result[]                // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности появления случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*mu*

[in] Логарифм математического ожидания (log\_mean).

*sigma*

[in] Логарифм среднеквадратического отклонения (log standard deviation).

*tail*

[in] Флаг расчета, если false, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если log\_mode=true, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomLognormal

Генерирует псевдослучайную величину, распределенную по логнормальному закону с параметрами *mu* и *sigma*. В случае ошибки возвращает [NaN](#).

```
double MathRandomLognormal(
    const double mu,           // логарифм математического ожидания (log mean)
    const double sigma,         // логарифм среднеквадратического отклонения (log standard deviation)
    int& error_code            // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по логнормальному закону с параметрами *mu* и *sigma*. В случае ошибки возвращает `false`. Аналог [rlnorm\(\)](#) в R.

```
double MathRandomLognormal(
    const double mu,           // логарифм математического ожидания (log mean)
    const double sigma,         // логарифм среднеквадратического отклонения (log standard deviation)
    const int data_count,       // количество необходимых данных
    double& result[]           // массив со значениями псевдослучайных величин
);
```

### Параметры

*mu*

[in] Логарифм математического ожидания (log\_mean).

*sigma*

[in] Логарифм среднеквадратического отклонения (log standard deviation).

*data\_count*

[in] Количество необходимых данных.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив со значениями псевдослучайных величин.

## MathMomentsLognormal

Рассчитывает теоретические численные значения первых 4 моментов логнормального распределения. Возвращает true, если расчет моментов произведен успешно, иначе false.

```
double MathMomentsLognormal(
    const double mu,           // логарифм математического ожидания (log mean)
    const double sigma,         // логарифм среднеквадратического отклонения (log standard deviation)
    double& mean,              // переменная для среднего значения
    double& variance,          // переменная для дисперсии
    double& skewness,           // переменная для коэффициента асимметрии
    double& kurtosis,           // переменная для коэффициента эксцесса
    int& error_code            // переменная для записи кода ошибки
);
```

### Параметры

*mu*

[in] Логарифм математического ожидания (log\_mean).

*sigma*

[in] Логарифм среднеквадратического отклонения (log standard deviation).

*mean*

[in] Переменная для среднего значения.

*variance*

[out] Переменная для дисперсии.

*skewness*

[out] Переменная для коэффициента асимметрии.

*kurtosis*

[out] Переменная для коэффициента эксцесса.

*error code*

[out] Переменная для записи кода ошибки.

### Возвращаемое значение

Возвращает true, если моменты успешно рассчитаны, иначе false.

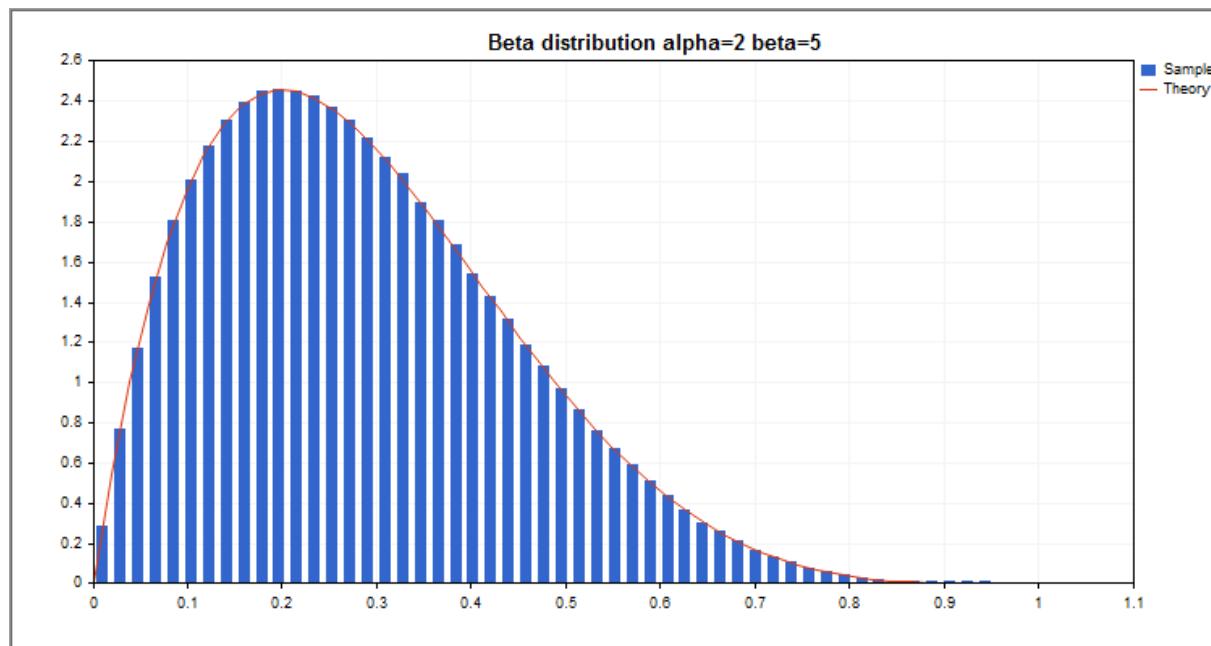
## Бета-распределение

В данном разделе представлены функции для работы с бета-распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по соответствующему закону. Бета-распределение описывается следующей формулой:

$$f_{Beta}(x | a, b) = \frac{1}{B(a, b)} x^{a-1} (1-x)^{b-1}$$

где:

- $x$  – значение случайной величины
- $a$  – первый параметр бета-распределения
- $b$  – второй параметр бета-распределения



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityBeta</a>	Рассчитывает плотность вероятности бета-распределения
<a href="#">MathCumulativeDistributionBeta</a>	Рассчитывает значение функции бета-распределения вероятностей
<a href="#">MathQuantileBeta</a>	Рассчитывает значение обратной функции бета-распределения для заданной вероятности
<a href="#">MathRandomBeta</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин,

	распределенных по закону бета-распределения
<a href="#">MathMomentsBeta</a>	Рассчитывает теоретические численные значения первых 4 моментов бета-распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Beta.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double alpha=2; // первый параметр бета-распределения (shape1)
input double beta=5; // второй параметр бета-распределения (shape2)
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
long chart=0;
string name="GraphicNormal";
int n=1000000; // количество значений в выборке
int ncells=51; // количество интервалов в гистограмме
double x[]; // центры интервалов гистограммы
double y[]; // количество значений из выборки, попавших в интервал
double data[]; // выборка случайных значений
double max,min; // максимальное и минимальное значения в выборке
//--- получим выборку из бета-распределения
MathRandomBeta(alpha,beta,n,data);
//--- рассчитаем данные для построения гистограммы
CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
double step;
GetMaxMinStepValues(max,min,step);
step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
double x2[];
double y2[];
MathSequence(min,max,step,x2);
MathProbabilityDensityBeta(x2,alpha,beta,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
```

```

double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<nccells; i++)
    y[i]/=k;
//--- ВЫВОДИМ ГРАФИКИ
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Beta distribution alpha=%G beta=%G",alpha,beta));
graphic.BackgroundMainSize(16);
//--- ПОСТРОИМ ВСЕ КРИВЫЕ
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- А ТЕПЕРЬ ПОСТРОИМ ТЕОРЕТИЧЕСКУЮ КРИВУЮ ПЛОТНОСТИ РАСПРЕДЕЛЕНИЯ
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- ПОСТРОИМ ВСЕ КРИВЫЕ
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
}

```

```
    return (true);
}

//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if ((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityBeta

Рассчитывает плотность вероятности бета-распределения с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityBeta(
    const double x,           // значение случайной величины
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается ln(p)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности бета-распределения с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityBeta(
    const double x,           // значение случайной величины
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности бета-распределения с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [dbeta\(\)](#) в R.

```
bool MathProbabilityDensityBeta(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // первый параметр бета-распределения (shape1)
    const double b,            // второй параметр бета-распределения (shape2)
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается ln(p)
    double& result[]          // массив для значения функции плотности вероятности
);
```

Рассчитывает плотность вероятности бета-распределения с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityBeta(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // первый параметр бета-распределения (shape1)
    const double b,            // второй параметр бета-распределения (shape2)
    double& result[]          // массив для значения функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*a*

[in] Первый параметр бета-распределения (shape 1).

*b*

[in] Второй параметр бета-распределения (shape 2)

*log\_mode*

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значения функции плотности вероятности.

## MathCumulativeDistributionBeta

Рассчитывает распределение вероятностей бета-распределения с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionBeta(
    const double x,           // значение случайной величины
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const bool tail,          // флаг расчета, если true, то рассчитывается вероятно
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то в
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей бета-распределения с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionBeta(
    const double x,           // значение случайной величины
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей бета-распределения с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [pbeta\(\)](#) в R.

```
bool MathCumulativeDistributionBeta(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // первый параметр бета-распределения (shape1)
    const double b,            // второй параметр бета-распределения (shape2)
    const bool tail,          // флаг расчета, если true, то рассчитывается вероятно
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то в
    double& result[]          // массив для значения функции вероятности
);
```

Рассчитывает распределение вероятностей бета-распределения с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathCumulativeDistributionBeta(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // первый параметр бета-распределения (shape1)
    const double b,            // второй параметр бета-распределения (shape2)
    double& result[]          // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*a*

[in] Первый параметр бета-распределения (shape 1).

*b*

[in] Второй параметр бета-распределения (shape 2)

*tail*

[in] Флаг расчета, если true, то рассчитывается вероятность того, что случайная величина не превысит x.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode*=true, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileBeta

Рассчитывает для вероятности *probability* значение обратной функции бета-распределения с параметрами *a* и *b*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileBeta(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // первый параметр бета-распределения (shape1)
    const double b,                     // второй параметр бета-распределения (shape2)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифмической вероятности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает для вероятности *probability* значение обратной функции бета-распределения с параметрами *a* и *b*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileBeta(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // первый параметр бета-распределения (shape1)
    const double b,                     // второй параметр бета-распределения (shape2)
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает для массива значений вероятности *probability[]* значения обратной функции бета-распределения с параметрами *a* и *b*. В случае ошибки возвращает `false`. Аналог [qbeta\(\)](#) в R.

```
double MathQuantileBeta(
    const double& probability[], // массив со значениями вероятностей случайной величины
    const double a,               // первый параметр бета-распределения (shape1)
    const double b,               // второй параметр бета-распределения (shape2)
    const bool tail,             // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,         // флаг расчета, если log_mode=true, то расчет ведется для логарифмической вероятности
    double& result[]            // массив со значениями квантилей
);
```

Рассчитывает для массива значений вероятности *probability[]* значения обратной функции бета-распределения с параметрами *a* и *b*. В случае ошибки возвращает `false`.

```
bool MathQuantileBeta(
    const double& probability[], // массив со значениями вероятностей случайной величины
    const double a,               // первый параметр бета-распределения (shape1)
    const double b,               // второй параметр бета-распределения (shape2)
    double& result[]            // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*a*

[in] Первый параметр бета-распределения (shape1).

*b*

[in] Второй параметр бета-распределения (shape2).

*tail*

[in] Флаг расчета, если lower\_tail=false, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если log\_mode=true, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomBeta

Генерирует псевдослучайную величину, распределенную по закону бета-распределения с параметрами *a* и *b*. В случае ошибки возвращает [NaN](#).

```
double MathRandomBeta(
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    int& error_code          // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону бета-распределения с параметрами *a* и *b*. В случае ошибки возвращает `false`. Аналог [rbeta\(\)](#) в R.

```
bool MathRandomBeta(
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const int data_count,     // количество необходимых данных
    double& result[]          // массив для получения псевдослучайных величин
);
```

### Параметры

*a*

[in] Первый параметр бета-распределения (shape1)

*b*

[in] Второй параметр бета-распределения (shape2).

*data\_count*

[in] Количество псевдослучайных значений, которые необходимо получить.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsBeta

Рассчитывает теоретические численные значения первых 4 моментов бета-распределения.

```
double MathMomentsBeta(
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    double& mean,            // переменная для среднего значения
    double& variance,        // переменная для дисперсии
    double& skewness,         // переменная для коэффициента асимметрии
    double& kurtosis,         // переменная для коэффициента эксцесса
    int& error_code          // переменная для кода ошибки
);
```

### Параметры

*a*

[in] Первый параметр бета-распределения (shape1).

*b*

[in] Второй параметр бета-распределения (shape2).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если моменты успешно рассчитаны, иначе false.

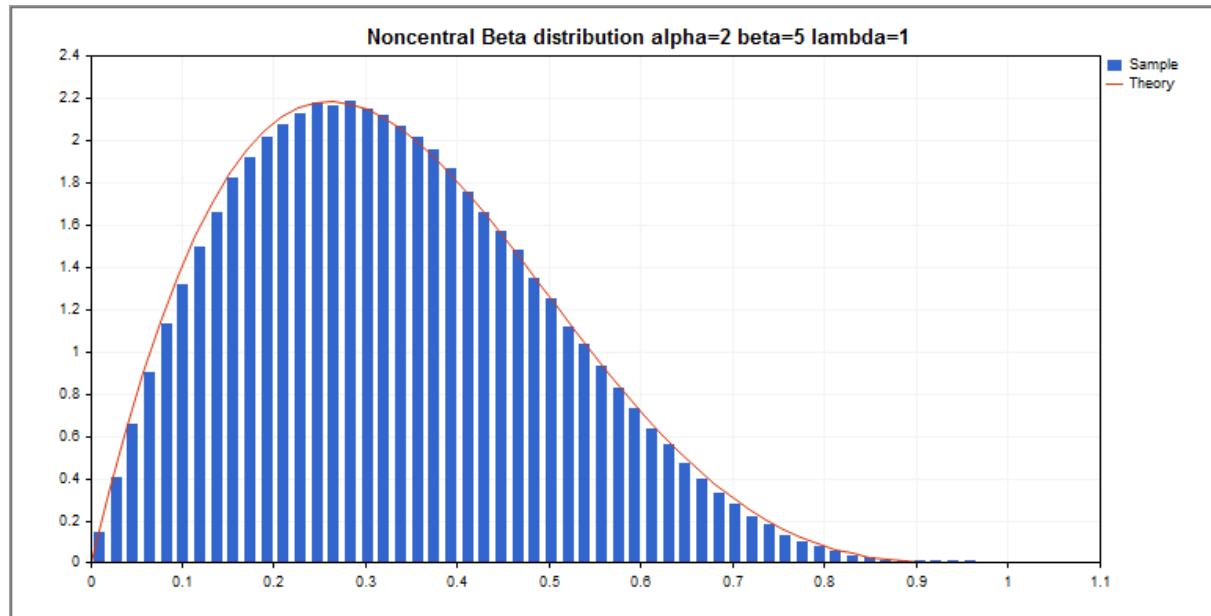
## Нецентральное бета-распределение

В данном разделе представлены функции для работы с нецентральным бета-распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по соответствующему закону. Нецентральное бета-распределение описывается следующей формулой:

$$f_{NoncentralBeta}(x|a,b,\lambda) = \sum_{r=0}^{\infty} e^{-\frac{\lambda}{2}} \frac{\left(\frac{\lambda}{2}\right)^r}{r!} \frac{x^{a+r-1} (1-x)^{b-1}}{B(a+r,b)}$$

где:

- $x$  — значение случайной величины
- $a$  — первый параметр бета-распределения
- $b$  — второй параметр бета-распределения
- $\lambda$  — параметр нецентральности



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityNoncentralBeta</a>	Рассчитывает плотность вероятности нецентрального бета-распределения
<a href="#">MathCumulativeDistributionNoncentralBeta</a>	Рассчитывает значение функции вероятностей нецентрального бета-распределения
<a href="#">MathQuantileNoncentralBeta</a>	Рассчитывает значение обратной функции нецентрального бета-распределения для заданной вероятности

<a href="#"><u>MathRandomNoncentralBeta</u></a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по закону нецентрального бета-распределения
<a href="#"><u>MathMomentsNoncentralBeta</u></a>	Рассчитывает теоретические численные значения первых 4 моментов нецентрального бета-распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralBeta.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs

//--- input parameters
input double a_par=2;           // первый параметр бета-распределения (shape1)
input double b_par=5;           // второй параметр бета-распределения (shape2)
input double l_par=1;           // параметр нецентральности (lambda)

//+-----+
//| Script program start function          |
//+-----+
void OnStart()
{
    //--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
    //--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
    //--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;                // количество значений в выборке
    int ncells=53;                // количество интервалов в гистограмме
    double x[];                  // центры интервалов гистограммы
    double y[];                  // количество значений из выборки, попавших в интервал
    double data[];               // выборка случайных значений
    double max,min;              // максимальное и минимальное значения в выборке

    //--- получим выборку из нецентрального бета-распределения
    MathRandomNoncentralBeta(a_par,b_par,l_par,n,data);
    //--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
    //--- получим границы последовательности и шаг для построения теоретической кривой
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);

    //--- получим теоретически рассчитанные данные на интервале [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
}
```

```

MathProbabilityDensityNoncentralBeta(x2,a_par,b_par,l_par,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<nCells; i++)
    y[i]/=k;
//--- выводим графики
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Noncentral Beta distribution alpha=%G beta=%G
                                         a_par,b_par,l_par"));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |+
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {

```

```
int ind=int((data[i]-minv)/width);
if(ind>=cells) ind=cells-1;
frequency[ind]++;
}
return (true);
}

//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
double range=MathAbs(maxv-minv);
int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
maxv=NormalizeDouble(maxv,degree);
minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
stepv=NormalizeDouble(MathPow(10,-degree),degree);
if((maxv-minv)/stepv<10)
stepv/=10.;

}
```

## MathProbabilityDensityNoncentralBeta

Рассчитывает плотность вероятности нецентрального бета-распределения с параметрами a, b и lambda для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNoncentralBeta(
    const double x,           // значение случайной величины
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const double lambda,      // параметр нецентральности
    const bool log_mode,     // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности нецентрального бета-распределения с параметрами a, b и lambda для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNoncentralBeta(
    const double x,           // значение случайной величины
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const double lambda,      // параметр нецентральности
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности нецентрального бета-распределения с параметрами a, b и lambda для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [dbeta\(\)](#) в R.

```
bool MathProbabilityDensityNoncentralBeta(
    const double& x[],        // массив со значениями случайной величины
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const double lambda,      // параметр нецентральности
    const bool log_mode,     // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]         // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности нецентрального бета-распределения с параметрами a, b и lambda для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathProbabilityDensityNoncentralBeta(
    const double& x[],        // массив со значениями случайной величины
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const double lambda,      // параметр нецентральности
    double& result[]         // массив для значений функции плотности вероятности
);
```

### Параметры

`x`

[in] Значение случайной величины.

`x[]`

[in] Массив со значениями случайной величины.

`a`

[in] Первый параметр бета-распределения (shape 1).

`b`

[in] Второй параметр бета-распределения (shape 2)

`lambda`

[in] Параметр нецентральности

`log_mode`

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

`error_code`

[out] Переменная для записи кода ошибки.

`result[]`

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionNoncentralBeta

Рассчитывает распределение вероятностей нецентрального бета-распределения с параметрами a, b и lambda для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionNoncentralBeta(
    const double x,           // значение случайной величины
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const double lambda,      // параметр нецентральности
    const bool tail,          // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается логарифм вероятности
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей нецентрального бета-распределения с параметрами a, b и lambda для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionNoncentralBeta(
    const double x,           // значение случайной величины
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const double lambda,      // параметр нецентральности
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей нецентрального бета-распределения с параметрами a, b и lambda для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [pbeta\(\)](#) в R.

```
bool MathCumulativeDistributionNoncentralBeta(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // первый параметр бета-распределения (shape1)
    const double b,            // второй параметр бета-распределения (shape2)
    const double lambda,       // параметр нецентральности
    const bool tail,           // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм вероятности
    double& result[]          // массив для значений функции вероятности
);
```

Рассчитывает распределение вероятностей нецентрального бета-распределения с параметрами a, b и lambda для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathCumulativeDistributionNoncentralBeta(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // первый параметр бета-распределения (shape1)
    const double b,            // второй параметр бета-распределения (shape2)
    const double lambda,       // параметр нецентральности
    double& result[]          // массив для значений функции вероятности
);
```

## Параметры

*x*

[in] Значение случайной величины.

*x* [ ]

[in] Массив со значениями случайной величины.

*a*

[in] Первый параметр бета-распределения (shape 1).

*b*

[in] Второй параметр бета-распределения (shape 2)

*lambda*

[in] Параметр нецентральности

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result* [ ]

[out] Массив для значений функции вероятности.

## MathQuantileNoncentralBeta

Рассчитывает значение обратной функции распределения вероятностей нецентрального бета-распределения с параметрами a, b и lambda для вероятности появления значения случайной величины *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileNoncentralBeta(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // первый параметр бета-распределения (shape1)
    const double b,                     // второй параметр бета-распределения (shape2)
    const double lambda,               // параметр нецентральности
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифма вероятности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции распределения вероятностей нецентрального бета-распределения с параметрами a, b и lambda для вероятности появления значения случайной величины *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileNoncentralBeta(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // первый параметр бета-распределения (shape1)
    const double b,                     // второй параметр бета-распределения (shape2)
    const double lambda,               // параметр нецентральности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции распределения вероятностей нецентрального бета-распределения с параметрами a, b и lambda для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*. Аналог [qbeta\(\)](#) в R.

```
double MathQuantileNoncentralBeta(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double a,                     // первый параметр бета-распределения (shape1)
    const double b,                     // второй параметр бета-распределения (shape2)
    const double lambda,               // параметр нецентральности
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability[i]
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифма вероятности
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции распределения вероятностей нецентрального бета-распределения с параметрами a, b и lambda для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*.

```
bool MathQuantileNoncentralBeta(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double a,                     // первый параметр бета-распределения (shape1)
    const double b,                     // второй параметр бета-распределения (shape2)
```

```
const double lambda,           // параметр нецентральности
          double& result[];    // массив со значениями квантилей
};
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*a*

[in] Первый параметр бета-распределения (shape1).

*b*

[in] Второй параметр бета-распределения (shape2).

*lambda*

[in] Параметр нецентральности.

*tail*

[in] Флаг расчета, если *false*, то расчет ведется для вероятности 1.0-*probability*.

*log\_mode*

[in] Флаг расчета, если *log\_mode=true*, то расчет ведется для вероятности Exp(*probability*).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomNoncentralBeta

Генерирует псевдослучайную величину, распределенную по закону нецентрального бета-распределения с параметрами a, b и lambda. В случае ошибки возвращает [NaN](#).

```
double MathRandomNoncentralBeta(
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const double lambda,      // параметр нецентральности
    int& error_code          // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону нецентрального бета-распределения с параметрами a, b и lambda. В случае ошибки возвращает false. Аналог [rbeta\(\)](#) в R.

```
bool MathRandomNoncentralBeta(
    const double a,           // первый параметр бета-распределения (shape1)
    const double b,           // второй параметр бета-распределения (shape2)
    const double lambda,      // параметр нецентральности
    const int data_count,     // количество необходимых данных
    double& result[]          // массив для получения псевдослучайных величин
);
```

### Параметры

*a*

[in] Первый параметр бета-распределения (shape1)

*b*

[in] Второй параметр бета-распределения (shape2).

*lambda*

[in] Параметр нецентральности

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsNoncentralBeta

Рассчитывает теоретические численные значения первых 4 моментов нецентрального бета-распределения с параметрами a, b и lambda.

```
double MathMomentsNoncentralBeta(
    const double a,                      // первый параметр бета-распределения (shape1)
    const double b,                      // второй параметр бета-распределения (shape2)
    const double lambda,                 // параметр нецентральности
    double& mean,                      // переменная для среднего значения
    double& variance,                  // переменная для дисперсии
    double& skewness,                  // переменная для коэффициента асимметрии
    double& kurtosis,                  // переменная для коэффициента эксцесса
    int& error_code                   // переменная для кода ошибки
);
```

### Параметры

*a*

[in] Первый параметр бета-распределения (shape1).

*b*

[in] Второй параметр бета-распределения (shape2).

*lambda*

[in] Параметр нецентральности

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если расчет моментов произведен успешно, иначе false.

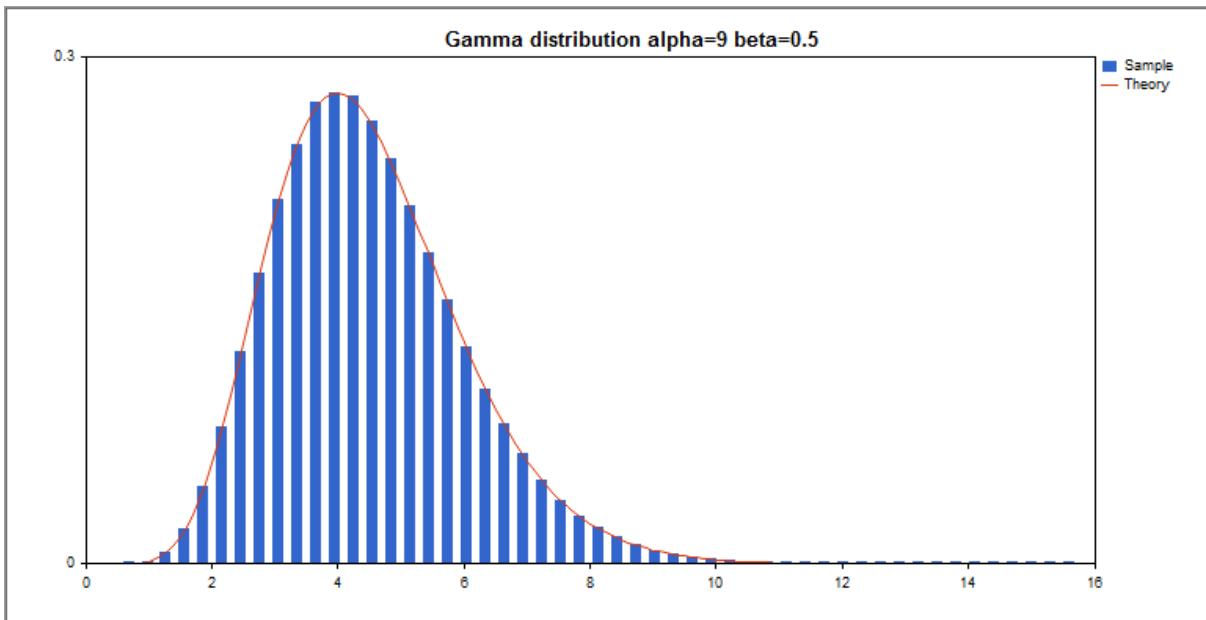
## Гамма-распределение

В данном разделе представлены функции для работы с гамма-распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по соответствующему закону. Гамма-распределение описывается следующей формулой:

$$f_{\text{Gamma}}(x | a, b) = \frac{1}{b^a \Gamma(a)} x^{a-1} e^{-\frac{x}{b}}$$

где:

- $x$  – значение случайной величины
- $a$  – первый параметр распределения
- $b$  – второй параметр распределения



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityGamma</a>	Рассчитывает плотность вероятности гамма-распределения
<a href="#">MathCumulativeDistributionGamma</a>	Рассчитывает значение функции гамма-распределения вероятностей
<a href="#">MathQuantileGamma</a>	Рассчитывает значение обратной функции гамма-распределения для заданной вероятности
<a href="#">MathRandomGamma</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин,

	распределенных по закону гамма-распределения
<a href="#">MathMomentsGamma</a>	Рассчитывает теоретические численные значения первых 4 моментов гамма-распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Gamma.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double alpha=9; // первый параметр гамма-распределения (shape)
input double beta=0.5; // второй параметр гамма-распределения (scale)
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
long chart=0;
string name="GraphicNormal";
int n=1000000; // количество значений в выборке
int ncells=51; // количество интервалов в гистограмме
double x[]; // центры интервалов гистограммы
double y[]; // количество значений из выборки, попавших в интервал
double data[]; // выборка случайных значений
double max,min; // максимальное и минимальное значения в выборке
//--- получим выборку из гамма-распределения
MathRandomGamma(alpha,beta,n,data);
//--- рассчитаем данные для построения гистограммы
CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
double step;
GetMaxMinStepValues(max,min,step);
step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
double x2[];
double y2[];
MathSequence(min,max,step,x2);
MathProbabilityDensityGamma(x2,alpha,beta,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
```

```

double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- ВЫВОДИМ ГРАФИКИ
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Gamma distribution alpha=%G beta=%G",alpha,beta));
graphic.BackgroundMainSize(16);
//--- отключим автомасштабирование оси Y
graphic.YAxis().AutoScale(false);
graphic.YAxis().Max(NormalizeDouble(theor_max,1));
graphic.YAxis().Min(0);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |+
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {

```

```
int ind=int((data[i]-minv)/width);
if(ind>=cells) ind=cells-1;
frequency[ind]++;
}
return (true);
}

//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
double range=MathAbs(maxv-minv);
int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
maxv=NormalizeDouble(maxv,degree);
minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
stepv=NormalizeDouble(MathPow(10,-degree),degree);
if((maxv-minv)/stepv<10)
stepv/=10.;

}
```

## MathProbabilityDensityGamma

Рассчитывает плотность вероятности гамма-распределения с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityGamma(
    const double x,           // значение случайной величины
    const double a,           // первый параметр распределения (shape)
    const double b,           // второй параметр распределения (scale)
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается ln(p(x))
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности гамма-распределения с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityGamma(
    const double x,           // значение случайной величины
    const double a,           // первый параметр распределения (shape)
    const double b,           // второй параметр распределения (scale)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности гамма-распределения с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [dgamma\(\)](#) в R.

```
bool MathProbabilityDensityGamma(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // первый параметр распределения (shape)
    const double b,            // второй параметр распределения (scale)
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается ln(p(x))
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности гамма-распределения с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityGamma(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // первый параметр распределения (shape)
    const double b,            // второй параметр распределения (scale)
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*a*

[in] Первый параметр распределения (shape).

*b*

[in] Второй параметр распределения (scale).

*log\_mode*

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionGamma

Рассчитывает гамма-распределение вероятностей с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionGamma(
    const double x,           // значение случайной величины
    const double a,           // первый параметр распределения (shape)
    const double b,           // второй параметр распределения (scale)
    const double tail,        // флаг расчета, если true, то рассчитывается вероятно
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то в
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает гамма-распределение вероятностей с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionGamma(
    const double x,           // значение случайной величины
    const double a,           // первый параметр распределения (shape)
    const double b,           // второй параметр распределения (scale)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает гамма-распределение вероятностей с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [pgamma\(\)](#) в R.

```
bool MathCumulativeDistributionGamma(
    const double& x[],         // массив со значениями случайной величины
    const double a,             // первый параметр распределения (shape)
    const double b,             // второй параметр распределения (scale)
    const double tail,          // флаг расчета, если true, то рассчитывается вероятно
    const bool log_mode,        // флаг расчета логарифма значения, если log_mode=true, то в
    double& result[]           // массив для значений функции вероятности
);
```

Рассчитывает гамма-распределение вероятностей с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathCumulativeDistributionGamma(
    const double& x[],         // массив со значениями случайной величины
    const double a,             // первый параметр распределения (shape)
    const double b,             // второй параметр распределения (scale)
    double& result[]           // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*a*

[in] Первый параметр распределения (shape).

*b*

[in] Второй параметр распределения (scale)

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileGamma

Рассчитывает значение обратной функции гамма-распределения с параметрами *a* и *b* для вероятности *probability*. В случае ошибки возвращает *NaN*.

```
double MathQuantileGamma(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // первый параметр распределения (shape)
    const double b,                     // второй параметр распределения (scale)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифмической вероятности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции гамма-распределения с параметрами *a* и *b* для вероятности *probability*. В случае ошибки возвращает *NaN*.

```
double MathQuantileGamma(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // первый параметр распределения (shape)
    const double b,                     // второй параметр распределения (scale)
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции гамма-распределения с параметрами *a* и *b* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*. Аналог [qgamma\(\)](#) в R.

```
double MathQuantileGamma(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double a,                     // первый параметр распределения (shape)
    const double b,                     // второй параметр распределения (scale)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифмической вероятности
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции гамма-распределения с параметрами *a* и *b* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*.

```
bool MathQuantileGamma(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double a,                     // первый параметр распределения (shape)
    const double b,                     // второй параметр распределения (scale)
    double& result[]                 // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*a*

[in] Первый параметр распределения (shape).

*b*

[in] Второй параметр распределения (scale).

*tail*

[in] Флаг расчета, если false, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если log\_mode=true, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomGamma

Генерирует псевдослучайную величину, распределенную по закону гамма-распределения с параметрами *a* и *b*. В случае ошибки возвращает [NaN](#).

```
double MathRandomGamma(
    const double a,           // первый параметр распределения (shape)
    const double b,           // второй параметр распределения (scale)
    int& error_code          // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону гамма-распределения с параметрами *a* и *b*. В случае ошибки возвращает `false`. Аналог [rgamma\(\)](#) в R.

```
bool MathRandomGamma(
    const double a,           // первый параметр распределения (shape)
    const double b,           // второй параметр распределения (scale)
    const int data_count,     // количество необходимых данных
    double& result[]          // массив со значениями псевдослучайных величин
);
```

### Параметры

*a*

[in] Первый параметр распределения (shape).

*b*

[in] Второй параметр распределения (scale).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsGamma

Рассчитывает теоретические численные значения первых 4 моментов гамма-распределения с параметрами *a*, *b*.

```
double MathMomentsGamma(
    const double a,           // первый параметр распределения (shape)
    const double b,           // второй параметр распределения (scale)
    double& mean,            // переменная для среднего значения
    double& variance,        // переменная для дисперсии
    double& skewness,         // переменная для коэффициента асимметрии
    double& kurtosis,         // переменная для коэффициента эксцесса
    int& error_code          // переменная для кода ошибки
);
```

### Параметры

*a*

[in] Первый параметр распределения (shape).

*b*

[in] Второй параметр распределения (scale).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает `true`, если расчет моментов произведен успешно, иначе `false`.

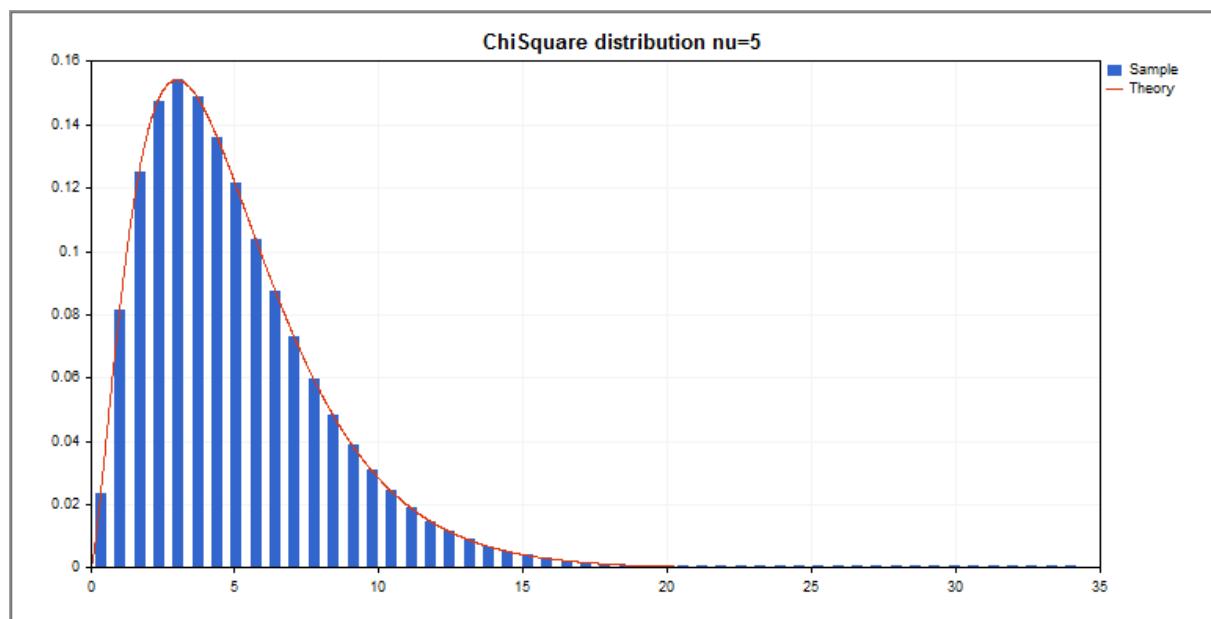
## Распределение хи-квадрат

В данном разделе представлены функции для работы с распределением хи-квадрат. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по соответствующему закону. Распределение хи-квадрат описывается следующей формулой:

$$f_{Chi-Square}(x | \nu) = \frac{x^{\frac{(\nu-2)}{2}} e^{-\frac{x}{2}}}{2^{\frac{\nu}{2}} \Gamma\left(\frac{\nu}{2}\right)}$$

где:

- $x$  — значение случайной величины
- $\nu$  — число степеней свободы



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityChiSquare</a>	Рассчитывает плотность вероятности распределения хи-квадрат
<a href="#">MathCumulativeDistributionChiSquare</a>	Рассчитывает значение функции распределения вероятностей хи-квадрат
<a href="#">MathQuantileChiSquare</a>	Рассчитывает значение обратной функции распределения хи-квадрат для заданной вероятности

<a href="#">MathRandomChiSquare</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по закону распределения хи-квадрат
<a href="#">MathMomentsChiSquare</a>	Рассчитывает теоретические численные значения первых 4 моментов распределения хи-квадрат

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\ChiSquare.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=5; // число степеней свободы
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
long chart=0;
string name="GraphicNormal";
int n=1000000; // количество значений в выборке
int ncells=51; // количество интервалов в гистограмме
double x[]; // центры интервалов гистограммы
double y[]; // количество значений из выборки, попавших в интервал
double data[]; // выборка случайных значений
double max,min; // максимальное и минимальное значения в выборке
//--- получим выборку из хи-квадрат распределения
MathRandomChiSquare(nu_par,n,data);
//--- рассчитаем данные для построения гистограммы
CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
double step;
GetMaxMinStepValues(max,min,step);
step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
double x2[];
double y2[];
MathSequence(min,max,step,x2);
MathProbabilityDensityChiSquare(x2,nu_par,false,y2);
//--- масштабируем
```

```

double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- выводим графики
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("ChiSquare distribution nu=%G ",nu_par));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |+
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
}

```

```
        }
        return (true);
    }
//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityChiSquare

Рассчитывает плотность вероятности распределения хи-квадрат с параметром nu для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityChiSquare(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается log(p)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности распределения хи-квадрат с параметром nu для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityChiSquare(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности распределения хи-квадрат с параметром nu для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [dchisq\(\)](#) в R.

```
bool MathProbabilityDensityChiSquare(
    const double& x[],        // массив со значениями случайной величины
    const double nu,           // параметр распределения (число степеней свободы)
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается log(p)
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности распределения хи-квадрат с параметром nu для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathProbabilityDensityChiSquare(
    const double& x[],        // массив со значениями случайной величины
    const double nu,           // параметр распределения (число степеней свободы)
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

x

[in] Значение случайной величины.

x[]

[in] Массив со значениями случайной величины.

nu

[in] Параметр распределения (число степеней свободы)

log\_mode

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

`error_code`

[out] Переменная для записи кода ошибки.

`result[]`

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionChiSquare

Рассчитывает распределение вероятностей хи-квадрат с параметром *nu* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionChiSquare(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const double tail,         // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,       // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей хи-квадрат с параметром *nu* для случайной величины *x*. В случае ошибки возвращает [NaN](#)

```
double MathCumulativeDistributionChiSquare(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей хи-квадрат с параметром *nu* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [pchisq\(\)](#) в R.

```
bool MathCumulativeDistributionChiSquare(
    const double& x[],        // массив со значениями случайной величины
    const double nu,           // параметр распределения (число степеней свободы)
    const double tail,          // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,        // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]           // массив для значений функции вероятности
);
```

Рассчитывает распределение вероятностей хи-квадрат с параметром *nu* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathCumulativeDistributionChiSquare(
    const double& x[],        // массив со значениями случайной величины
    const double nu,           // параметр распределения (число степеней свободы)
    double& result[]           // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*nu*

[in] Параметр распределения (число степеней свободы).

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит x.

*log\_mode*

[in] Флаг расчета логарифма значения. Если log\_mode=true, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileChiSquare

Рассчитывает значение обратной функции распределения вероятностей хи-квадрат для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileChiSquare(
    const double probability,           // значение вероятности появления случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифмической вероятности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции распределения вероятностей хи-квадрат для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileChiSquare(
    const double probability,           // значение вероятности появления случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции распределения вероятностей хи-квадрат для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`. Аналог [qchisq\(\)](#) в R.

```
double MathQuantileChiSquare(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифмической вероятности
    double& result[]                // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции распределения вероятностей хи-квадрат для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`.

```
bool MathQuantileChiSquare(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    double& result[]                // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*nu*

[in] Параметр распределения (число степеней свободы).

*tail*

[in] Флаг расчета, если false, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если log\_mode=true, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomChiSquare

Генерирует псевдослучайную величину, распределенную по закону распределения хи-квадрат с параметром nu. В случае ошибки возвращает [NaN](#).

```
double MathRandomChiSquare(
    const double nu,           // параметр распределения (число степеней свободы)
    int& error_code            // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону распределения хи-квадрат с параметром nu. В случае ошибки возвращает `false`. Аналог [rchisq\(\)](#) в R.

```
bool MathRandomChiSquare(
    const double nu,           // параметр распределения (число степеней свободы)
    const int data_count,       // количество необходимых данных
    double& result[]           // массив со значениями псевдослучайных величин
);
```

### Параметры

*nu*

[in] Параметр распределения (число степеней свободы).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsChiSquare

Рассчитывает теоретические численные значения первых 4 моментов распределения хи-квадрат с параметром nu.

```
double MathMomentsChiSquare(
    const double nu,           // параметр распределения (число степеней свободы)
    double& mean,             // переменная для среднего значения
    double& variance,         // переменная для дисперсии
    double& skewness,          // переменная для коэффициента асимметрии
    double& kurtosis,          // переменная для коэффициента эксцесса
    int& error_code           // переменная для кода ошибки
);
```

### Параметры

*nu*

[in] Параметр распределения (число степеней свободы).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если расчет моментов произведен успешно, иначе false.

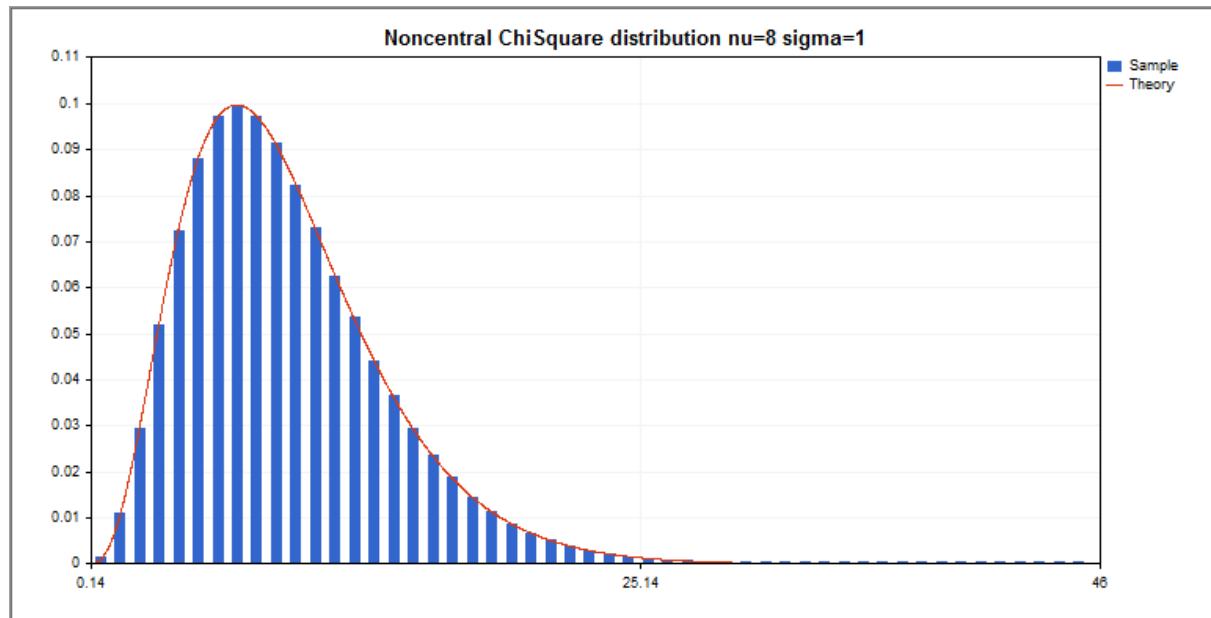
## Нецентральное распределение хи-квадрат

В данном разделе представлены функции для работы с нецентральным распределением хи-квадрат. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по соответствующему закону. Нецентральное распределение хи-квадрат описывается следующей формулой:

$$f_{NoncentralChiSquare}(x|v,\sigma) = \frac{1}{2^{\frac{v}{2}}\Gamma\left(\frac{1}{2}\right)} x^{\frac{v}{2}-1} e^{-\frac{(x+\sigma)^2}{2}} \sum_{r=0}^{\infty} \frac{(\lambda x)^r}{(2r)!} \frac{\Gamma\left(\frac{1}{2}+r\right)}{\Gamma\left(\frac{v}{2}+r\right)}$$

где:

- $x$  — значение случайной величины
- $v$  — число степеней свободы
- $\sigma$  — параметр нецентральности



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityNoncentralChiSquare</a>	Рассчитывает плотность вероятности нецентрального распределения хи-квадрат
<a href="#">MathCumulativeDistributionNoncentralChiSquare</a>	Рассчитывает значение функции нецентрального распределения вероятностей хи-квадрат
<a href="#">MathQuantileNoncentralChiSquare</a>	Рассчитывает значение обратной функции нецентрального распределения хи-квадрат для заданной вероятности

<a href="#">MathRandomNoncentralChiSquare</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по закону нецентрального распределения хи-квадрат
<a href="#">MathMomentsNoncentralChiSquare</a>	Рассчитывает теоретические численные значения первых 4 моментов нецентрального распределения хи-квадрат

Пример:

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralChiSquare.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=8;      // число степеней свободы
input double si_par=1;        // параметр нецентральности
//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;           // количество значений в выборке
    int ncells=51;            // количество интервалов в гистограмме
    double x[];               // центры интервалов гистограммы
    double y[];               // количество значений из выборки, попавших в интервал
    double data[];             // выборка случайных значений
    double max,min;           // максимальное и минимальное значения в выборке
//--- получим выборку из нецентрального хи-квадрат распределения
    MathRandomNoncentralChiSquare(nu_par,si_par,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityNoncentralChiSquare(x2,nu_par,si_par,false,y2);
```

```

//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- выводим графики
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Noncentral ChiSquare distribution nu=%G sigma=%G",nu,sigma));
graphic.BackgroundMainSize(16);
//--- отключим автомасштабирование оси X
graphic.XAxis().AutoScale(false);
graphic.XAxis().Max(NormalizeDouble(max,0));
graphic.XAxis().Min(min);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
}

```

```
for(int i=0; i<size; i++)
{
    int ind=int((data[i]-minv)/width);
    if(ind>=cells) ind=cells-1;
    frequency[ind]++;
}
return (true);
}

//+-----+
//| Calculates values for sequence generation |
//+-----+

void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityNoncentralChiSquare

Рассчитывает плотность вероятности нецентрального распределения хи-квадрат с параметрами *nu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNoncentralChiSquare(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const double sigma,        // параметр нецентральности
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности нецентрального распределения хи-квадрат с параметрами *nu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNoncentralChiSquare(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const double sigma,        // параметр нецентральности
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности нецентрального распределения хи-квадрат с параметрами *nu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [dchisq\(\)](#) в R.

```
bool MathProbabilityDensityNoncentralChiSquare(
    const double& x[],         // массив со значениями случайной величины
    const double nu,            // параметр распределения (число степеней свободы)
    const double sigma,         // параметр нецентральности
    const bool log_mode,        // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]           // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности нецентрального распределения хи-квадрат с параметром *nu* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityNoncentralChiSquare(
    const double& x[],         // массив со значениями случайной величины
    const double nu,            // параметр распределения (число степеней свободы)
    const double sigma,         // параметр нецентральности
    double& result[]           // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*nui*

[in] Параметр распределения (число степеней свободы).

*sigma*

[in] Параметр нецентральности.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode*=true, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionNoncentralChiSquare

Рассчитывает распределение вероятностей нецентрального распределения хи-квадрат с параметрами nu и sigma для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionNoncentralChiSquare(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const double sigma,        // параметр нецентральности
    const bool tail,          // флаг расчета, если lower_tail=true, то рассчитывается
    const bool log_mode,       // расчет логарифма значения, если log_mode=true, то в
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей нецентрального распределения хи-квадрат с параметрами nu и sigma для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionNoncentralChiSquare(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const double sigma,        // параметр нецентральности
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей нецентрального распределения хи-квадрат с параметрами nu и sigma для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [pchisq\(\)](#) в R.

```
bool MathCumulativeDistributionNoncentralChiSquare(
    const double& x[],         // массив со значениями случайной величины
    const double nu,            // параметр распределения (число степеней свободы)
    const double sigma,         // параметр нецентральности
    const bool tail,           // флаг расчета, если lower_tail=true, то рассчитывается
    const bool log_mode,        // флаг расчета логарифма значения, если log_mode=true, то в
    double& result[]           // массив для значений функции вероятности
);
```

Рассчитывает распределение вероятностей нецентрального распределения хи-квадрат с параметрами nu и sigma для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathCumulativeDistributionNoncentralChiSquare(
    const double& x[],         // массив со значениями случайной величины
    const double nu,            // параметр распределения (число степеней свободы)
    const double sigma,         // параметр нецентральности
    double& result[]           // массив для значений функции вероятности
);
```

### Параметры

x

[in] Значение случайной величины.

*x* []

[in] Массив со значениями случайной величины.

*nu*

[in] Параметр распределения (число степеней свободы).

*sigma*

[in] Параметр нецентральности.

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode*=true, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result*[]

[out] Массив для значений функции вероятности.

## MathQuantileNoncentralChiSquare

Рассчитывает значение обратной функции нецентрального распределения хи-квадрат с параметрами *nu* и *sigma* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileNoncentralChiSquare(
    const double probability,           // значение вероятности появления случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const double sigma,                // параметр нецентральности
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции нецентрального распределения хи-квадрат с параметрами *nu* и *sigma* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileNoncentralChiSquare(
    const double probability,           // значение вероятности появления случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const double sigma,                // параметр нецентральности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции нецентрального распределения хи-квадрат с параметрами *nu* и *sigma* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*. Аналог [qchisq\(\)](#) в R.

```
double MathQuantileNoncentralChiSquare(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const double sigma,                // параметр нецентральности
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции нецентрального распределения вероятностей хи-квадрат для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*.

```
bool MathQuantileNoncentralChiSquare(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const double sigma,                // параметр нецентральности
    double& result[]                 // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*nu*

[in] Параметр распределения (число степеней свободы).

*sigma*

[in] Параметр нецентральности.

*tail*

[in] Флаг расчета, если `false`, то расчет ведется для вероятности  $1.0 - \text{probability}$ .

*log\_mode*

[in] Флаг расчета, если `log_mode=true`, то расчет ведется для вероятности  $\text{Exp}(\text{probability})$ .

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomNoncentralChiSquare

Генерирует псевдослучайную величину, распределенную по закону нецентрального распределения хи-квадрат с параметрами nu и sigma. В случае ошибки возвращает [NaN](#).

```
double MathRandomNoncentralChiSquare(
    const double nu,           // параметр распределения (число степеней свободы)
    const double sigma,         // параметр нецентральности
    int& error_code            // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону нецентрального распределения хи-квадрат с параметрами nu и sigma. В случае ошибки возвращает false. Аналог [rchisq\(\)](#) в R.

```
bool MathRandomNoncentralChiSquare(
    const double nu,           // параметр распределения (число степеней свободы)
    const double sigma,         // параметр нецентральности
    const int data_count,       // количество необходимых данных
    double& result[]           // массив со значениями псевдослучайных величин
);
```

### Параметры

*nu*

[in] Параметр распределения (число степеней свободы).

*sigma*

[in] Параметр нецентральности.

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsNoncentralChiSquare

Рассчитывает теоретические численные значения первых 4 моментов нецентрального распределения хи-квадрат с параметрами *nu* и *sigma*.

```
double MathMomentsNoncentralChiSquare(
    const double nu,           // параметр распределения (число степеней свободы)
    const double sigma,         // параметр нецентральности
    double& mean,              // переменная для среднего значения
    double& variance,          // переменная для дисперсии
    double& skewness,           // переменная для коэффициента асимметрии
    double& kurtosis,           // переменная для коэффициента эксцесса
    int& error_code            // переменная для кода ошибки
);
```

### Параметры

*nu*

[in] Параметр распределения (число степеней свободы).

*sigma*

[in] Параметр нецентральности.

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает `true`, если расчет моментов произведен успешно, иначе `false`.

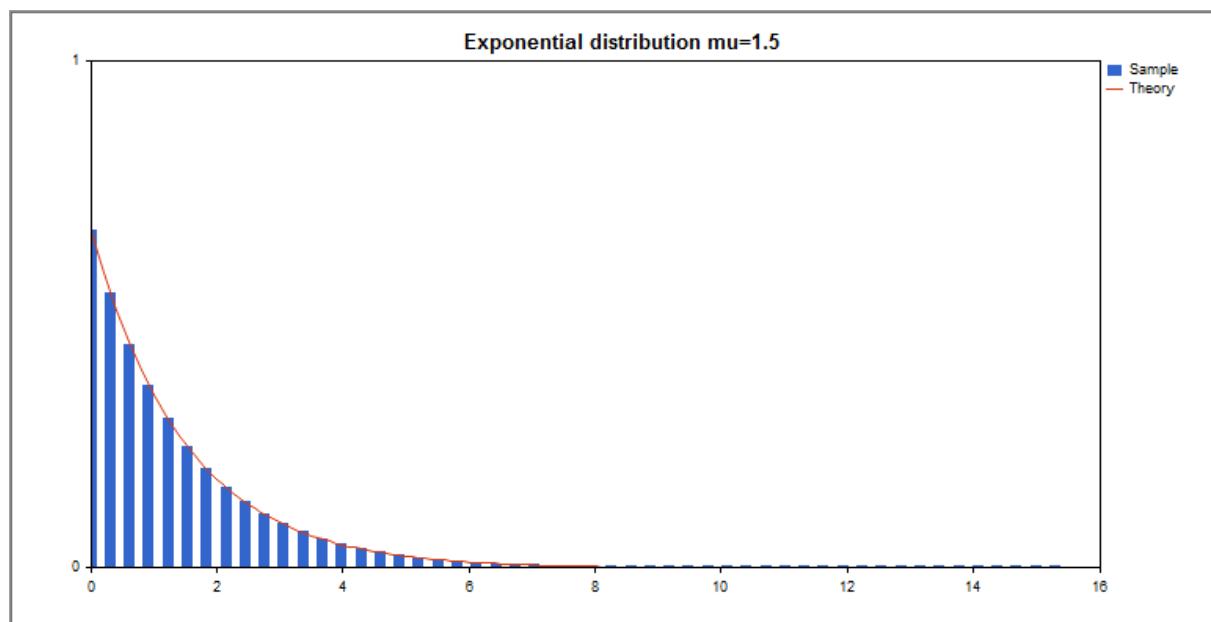
## Экспоненциальное

В данном разделе представлены функции для работы с экспоненциальным распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по экспоненциальному закону. Экспоненциальное распределение описывается следующей формулой:

$$f_{\text{Exponential}}(x | \mu) = \frac{1}{\mu} e^{-\frac{x}{\mu}}$$

где:

- $x$  — значение случайной величины
- $\mu$  — математическое ожидание



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityExponential</a>	Рассчитывает плотность вероятности экспоненциального распределения
<a href="#">MathCumulativeDistributionExponential</a>	Рассчитывает значение функции экспоненциального распределения вероятностей
<a href="#">MathQuantileExponential</a>	Рассчитывает значение обратной функции экспоненциального распределения для заданной вероятности
<a href="#">MathRandomExponential</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин,

	распределенных по экспоненциальному закону
<a href="#">MathMomentsExponential</a>	Рассчитывает теоретические численные значения первых 4 моментов экспоненциального распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Exponential.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mu_par=1.5; // число степеней свободы
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
MathSrand(GetTickCount());
//--- сгенерируем выборку случайной величины
long chart=0;
string name="GraphicNormal";
int n=1000000; // количество значений в выборке
int ncells=51; // количество интервалов в гистограмме
double x[]; // центры интервалов гистограммы
double y[]; // количество значений из выборки, попавших в интервал
double data[]; // выборка случайных значений
double max,min; // максимальное и минимальное значения в выборке
//--- получим выборку из экспоненциального распределения
MathRandomExponential(mu_par,n,data);
//--- рассчитаем данные для построения гистограммы
CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
double step;
GetMaxMinStepValues(max,min,step);
step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
double x2[];
double y2[];
MathSequence(min,max,step,x2);
MathProbabilityDensityExponential(x2,mu_par,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
```

```

        double k=sample_max/theor_max;
        for(int i=0; i<ncells; i++)
            y[i]/=k;
        //--- ВЫВОДИМ ГРАФИКИ
        CGraphic graphic;
        if(ObjectFind(chart,name)<0)
            graphic.Create(chart,name,0,0,0,780,380);
        else
            graphic.Attach(chart,name);
        graphic.BackgroundMain(StringFormat("Exponential distribution mu=%G ",mu_par));
        graphic.BackgroundMainSize(16);
        //--- plot all curves
        graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
        //--- а теперь построим теоретическую кривую плотности распределения
        graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
        graphic.CurvePlotAll();
        //--- plot all curves
        graphic.Update();
    }
//+-----+
//| Calculate frequencies for data set |+
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
    //--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+i*width;
        frequency[i]=0;
    }
    //--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```

```
}

//+-----+
//| Calculates values for sequence generation |
//+-----+

void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityExponential

Рассчитывает плотность вероятности экспоненциального распределения с параметром *mu* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityExponential(
    const double x,           // значение случайной величины
    const double mu,          // параметр распределения (математическое ожидание)
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается ln(p(x))
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности экспоненциального распределения с параметром *mu* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityExponential(
    const double x,           // значение случайной величины
    const double mu,          // параметр распределения (математическое ожидание)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности экспоненциального распределения с параметром *mu* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [dexp\(\)](#) в R.

```
bool MathProbabilityDensityExponential(
    const double& x[],        // массив со значениями случайной величины
    const double mu,           // параметр распределения (математическое ожидание)
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается ln(p(x))
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности экспоненциального распределения с параметром *mu* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityExponential(
    const double& x[],        // массив со значениями случайной величины
    const double mu,           // параметр распределения (математическое ожидание)
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*mu*

[in] Параметр распределения (математическое ожидание)

*log\_mode*

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

`error_code`

[out] Переменная для записи кода ошибки.

`result[]`

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionExponential

Рассчитывает экспоненциальное распределение вероятностей с параметром *mu* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionExponential(
    const double x,           // значение случайной величины
    const double mu,          // параметр распределения (математическое ожидание)
    const double tail,         // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,       // расчет логарифма значения, если log_mode=true, то возвращается ln(p)
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает экспоненциальное распределение вероятностей с параметром *mu* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionExponential(
    const double x,           // значение случайной величины
    const double mu,          // параметр распределения (математическое ожидание)
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает экспоненциальное распределение вероятностей с параметром *mu* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [pexp\(\)](#) в R.

```
bool MathCumulativeDistributionExponential(
    const double& x[],        // массив со значениями случайной величины
    const double mu,           // параметр распределения (математическое ожидание)
    const double tail,          // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,        // флаг расчета логарифма значения, если log_mode=true, то возвращается ln(p)
    double& result[]           // массив для значений функции вероятности
);
```

Рассчитывает экспоненциальное распределение вероятностей с параметром *mu* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathCumulativeDistributionExponential(
    const double& x[],        // массив со значениями случайной величины
    const double mu,           // параметр распределения (математическое ожидание)
    double& result[]           // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*mu*

[in] Параметр распределения (математическое ожидание).

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит x.

*log\_mode*

[in] Флаг расчета логарифма значения. Если log\_mode=true, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileExponential

Рассчитывает значение обратной функции экспоненциального распределения вероятностей с параметром *mu* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileExponential(
    const double probability,           // значение вероятности появления случайной величины
    const double mu,                   // параметр распределения (математическое ожидание)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции экспоненциального распределения вероятностей с параметром *mu* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileExponential(
    const double probability,           // значение вероятности появления случайной величины
    const double mu,                   // параметр распределения (математическое ожидание)
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции экспоненциального распределения вероятностей с параметром *mu* для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`. Аналог [qexp\(\)](#) в R.

```
double MathQuantileExponential(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double mu,                   // параметр распределения (математическое ожидание)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции экспоненциального распределения вероятностей с параметром *mu* для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`.

```
bool MathQuantileExponential(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double mu,                   // параметр распределения (математическое ожидание)
    double& result[]                 // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*mu*

[in] Параметр распределения (математическое ожидание).

*tail*

[in] Флаг расчета, если `false`, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если `log_mode=true`, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomExponential

Генерирует псевдослучайную величину, распределенную по закону экспоненциального распределения с параметром *mu*. В случае ошибки возвращает [NaN](#).

```
double MathRandomExponential(
    const double mu,           // параметр распределения (математическое ожидание)
    int& error_code            // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону экспоненциального распределения с параметром *mu*. В случае ошибки возвращает `false`. Аналог [rexp\(\)](#) в R.

```
bool MathRandomExponential(
    const double mu,           // параметр распределения (математическое ожидание)
    const int data_count,       // количество необходимых данных
    double& result[]           // массив со значениями псевдослучайных величин
);
```

### Параметры

*mu*

[in] Параметр распределения (математическое ожидание).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsExponential

Рассчитывает теоретические численные значения первых 4 моментов экспоненциального распределения с параметром *mu*.

```
double MathMomentsExponential(
    const double mu, // параметр распределения (математическое ожидание)
    double& mean, // переменная для среднего значения
    double& variance, // переменная для дисперсии
    double& skewness, // переменная для коэффициента асимметрии
    double& kurtosis, // переменная для коэффициента эксцесса
    int& error_code // переменная для кода ошибки
);
```

### Параметры

*mu*

[in] Параметр распределения (математическое ожидание).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает `true`, если расчет моментов произведен успешно, иначе `false`.

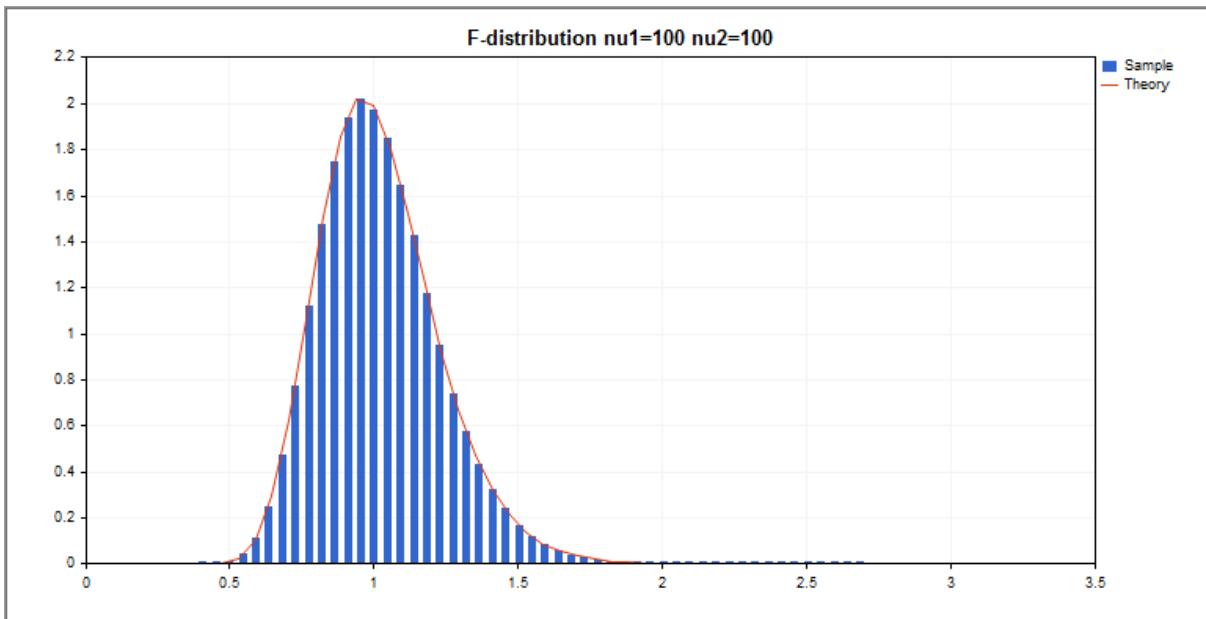
## F-распределение

В данном разделе представлены функции для работы с F-распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по закону Фишера. F-распределение описывается следующей формулой:

$$f_F(x|\nu_1, \nu_2) = \frac{\Gamma\left(\frac{\nu_1 + \nu_2}{2}\right)}{\Gamma\left(\frac{\nu_1}{2}\right)\Gamma\left(\frac{\nu_2}{2}\right)} \left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_1}{2}} \frac{x^{\frac{\nu_1 - 2}{2}}}{\left(1 + \left(\frac{\nu_1}{\nu_2}\right)x\right)^{\frac{\nu_1 + \nu_2}{2}}}$$

где:

- $x$  — значение случайной величины
- $\nu_1$  — первый параметр распределения (число степеней свободы)
- $\nu_2$  — второй параметр распределения (число степеней свободы)



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityF</a>	Рассчитывает плотность вероятности F-распределения
<a href="#">MathCumulativeDistributionF</a>	Рассчитывает значение функции F-распределения вероятностей
<a href="#">MathQuantileF</a>	Рассчитывает значение обратной функции F-распределения для заданной вероятности
<a href="#">MathRandomF</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин,

	распределенных по закону Фишера
<u>MathMomentsF</u>	Рассчитывает теоретические численные значения первых 4 моментов распределения Фишера

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\F.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_1=100; // первое число степеней свободы
input double nu_2=100; // второе число степеней свободы
//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
MathSrand(GetTickCount());
//--- сгенерируем выборку случайной величины
long chart=0;
string name="GraphicNormal";
int n=1000000; // количество значений в выборке
int ncells=51; // количество интервалов в гистограмме
double x[]; // центры интервалов гистограммы
double y[]; // количество значений из выборки, попавших в интервал
double data[]; // выборка случайных значений
double max,min; // максимальное и минимальное значения в выборке
//--- получим выборку из F-распределения
MathRandomF(nu_1,nu_2,n,data);
//--- рассчитаем данные для построения гистограммы
CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
double step;
GetMaxMinStepValues(max,min,step);
step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
double x2[];
double y2[];
MathSequence(min,max,step,x2);
MathProbabilityDensityF(x2,nu_1,nu_2,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
```

```

        double k=sample_max/theor_max;
        for(int i=0; i<ncells; i++)
            y[i]/=k;
        //--- ВЫВОДИМ ГРАФИКИ
        CGraphic graphic;
        if(ObjectFind(chart,name)<0)
            graphic.Create(chart,name,0,0,0,780,380);
        else
            graphic.Attach(chart,name);
        graphic.BackgroundMain(StringFormat("F-distribution nu1=%G nu2=%G",nu_1,nu_2));
        graphic.BackgroundMainSize(16);
        //--- plot all curves
        graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(4);
        //--- а теперь построим теоретическую кривую плотности распределения
        graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
        graphic.CurvePlotAll();
        //--- plot all curves
        graphic.Update();
    }
//+-----+
//| Calculate frequencies for data set |+
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
    //--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
    //--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```

```
}

//+-----+
//| Calculates values for sequence generation |
//+-----+

void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityF

Рассчитывает плотность вероятности F-распределения Фишера с параметрами nu1 и nu2 для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityF(
    const double x,           // значение случайной величины
    const double nu1,          // первый параметр распределения (число степеней свободы)
    const double nu2,          // второй параметр распределения (число степеней свободы)
    const bool log_mode,       // расчет логарифма значения, если log_mode=true, то возвращается ln(p)
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности F-распределения Фишера с параметрами nu1 и nu2 для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityF(
    const double x,           // значение случайной величины
    const double nu1,          // первый параметр распределения (число степеней свободы)
    const double nu2,          // второй параметр распределения (число степеней свободы)
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности F-распределения Фишера с параметрами nu1 и nu2 для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [df\(\)](#) в R.

```
bool MathProbabilityDensityF(
    const double& x[],         // массив со значениями случайной величины
    const double nu1,            // первый параметр распределения (число степеней свободы)
    const double nu2,            // второй параметр распределения (число степеней свободы)
    const bool log_mode,         // флаг расчета логарифма значения, если log_mode=true, то возвращается ln(p)
    double& result[]            // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности F-распределения Фишера с параметрами nu1 и nu2 для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathProbabilityDensityF(
    const double& x[],         // массив со значениями случайной величины
    const double nu1,            // первый параметр распределения (число степеней свободы)
    const double nu2,            // второй параметр распределения (число степеней свободы)
    double& result[]            // массив для значений функции плотности вероятности
);
```

### Параметры

x

[in] Значение случайной величины.

x[]

[in] Массив со значениями случайной величины.

*nu1*

[in] Первый параметр распределения (число степеней свободы).

*nu2*

[in] Второй параметр распределения (число степеней свободы).

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionF

Рассчитывает распределение вероятностей по закону F-распределения Фишера с параметрами nu1 и nu2 для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionF(
    const double x,           // значение случайной величины
    const double nu1,          // первый параметр распределения (число степеней свободы)
    const double nu2,          // второй параметр распределения (число степеней свободы)
    const double tail,         // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,       // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей по закону F-распределения Фишера с параметрами nu1 и nu2 для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionF(
    const double x,           // значение случайной величины
    const double nu1,          // первый параметр распределения (число степеней свободы)
    const double nu2,          // второй параметр распределения (число степеней свободы)
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей по закону F-распределения Фишера с параметрами nu1 и nu2 для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [pf\(\)](#) в R.

```
bool MathCumulativeDistributionF(
    const double& x[],          // массив со значениями случайной величины
    const double nu1,             // первый параметр распределения (число степеней свободы)
    const double nu2,             // второй параметр распределения (число степеней свободы)
    const double tail,            // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,          // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]             // массив для значений функции вероятности
);
```

Рассчитывает распределение вероятностей по закону F-распределения Фишера с параметрами nu1 и nu2 для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathCumulativeDistributionF(
    const double& x[],          // массив со значениями случайной величины
    const double nu1,             // первый параметр распределения (число степеней свободы)
    const double nu2,             // второй параметр распределения (число степеней свободы)
    double& result[]             // массив для значений функции вероятности
);
```

### Параметры

x

[in] Значение случайной величины.

x[]

[in] Массив со значениями случайной величины.

*nu1*

[in] Первый параметр распределения (число степеней свободы).

*nu2*

[in] Второй параметр распределения (число степеней свободы).

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileF

Рассчитывает значение обратной функции F-распределения вероятностей с параметрами nu1 и nu2 для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileF(
    const double probability,           // значение вероятности появления случайной величины
    const double nu1,                  // первый параметр распределения (число степеней свободы)
    const double nu2,                  // второй параметр распределения (число степеней свободы)
    const bool tail,                 // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,             // флаг расчета, если log_mode=true, то расчет ведется для логарифмической вероятности
    int& error_code                // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции F-распределения вероятностей с параметрами nu1 и nu2 для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileF(
    const double probability,           // значение вероятности появления случайной величины
    const double nu1,                  // первый параметр распределения (число степеней свободы)
    const double nu2,                  // второй параметр распределения (число степеней свободы)
    int& error_code                // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции F-распределения вероятностей с параметрами nu1 и nu2 для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`. Аналог [qf\(\)](#) в R.

```
double MathQuantileF(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double nu1,                  // первый параметр распределения (число степеней свободы)
    const double nu2,                  // второй параметр распределения (число степеней свободы)
    const bool tail,                 // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,             // флаг расчета, если log_mode=true, то расчет ведется для логарифмической вероятности
    double& result[]                // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции F-распределения вероятностей с параметрами nu1 и nu2 для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`.

```
bool MathQuantileF(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double nu1,                  // первый параметр распределения (число степеней свободы)
    const double nu2,                  // второй параметр распределения (число степеней свободы)
    double& result[]                // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*nu1*

[in] Первый параметр распределения (число степеней свободы).

*nu2*

[in] Второй параметр распределения (число степеней свободы).

*tail*

[in] Флаг расчета, если `lower_tail=false`, то расчет ведется для вероятности  $1.0 - \text{probability}$ .

*log\_mode*

[in] Флаг расчета, если `log_mode=true`, то расчет ведется для вероятности  $\text{Exp}(\text{probability})$ .

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomF

Генерирует псевдослучайную величину, распределенную по закону F-распределения Фишера с параметрами nu1 и nu2. В случае ошибки возвращает `NaN`.

```
double MathRandomF(
    const double nu1,           // первый параметр распределения (число степеней свободы)
    const double nu2,           // второй параметр распределения (число степеней свободы)
    int& error_code            // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, по закону F-распределения Фишера с параметрами nu1 и nu2. В случае ошибки возвращает `false`. Аналог `rf()` в R.

```
bool MathRandomF(
    const double nu1,           // первый параметр распределения (число степеней свободы)
    const double nu2,           // второй параметр распределения (число степеней свободы)
    const int data_count,       // количество необходимых данных
    double& result[]           // массив со значениями псевдослучайных величин
);
```

### Параметры

*nu1*

[in] Первый параметр распределения (число степеней свободы).

*nu2*

[in] Второй параметр распределения (число степеней свободы).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsF

Рассчитывает теоретические численные значения первых 4 моментов F-распределения Фишера с параметрами nu1 и nu2.

```
double MathMomentsF(
    const double nu1,           // первый параметр распределения (число степеней свободы)
    const double nu2,           // второй параметр распределения (число степеней свободы)
    double& mean,              // переменная для среднего значения
    double& variance,          // переменная для дисперсии
    double& skewness,           // переменная для коэффициента асимметрии
    double& kurtosis,           // переменная для коэффициента эксцесса
    int& error_code            // переменная для кода ошибки
);
```

### Параметры

*nu1*

[in] Первый параметр распределения (число степеней свободы).

*nu2*

[in] Второй параметр распределения (число степеней свободы).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если расчет моментов произведен успешно, иначе false.

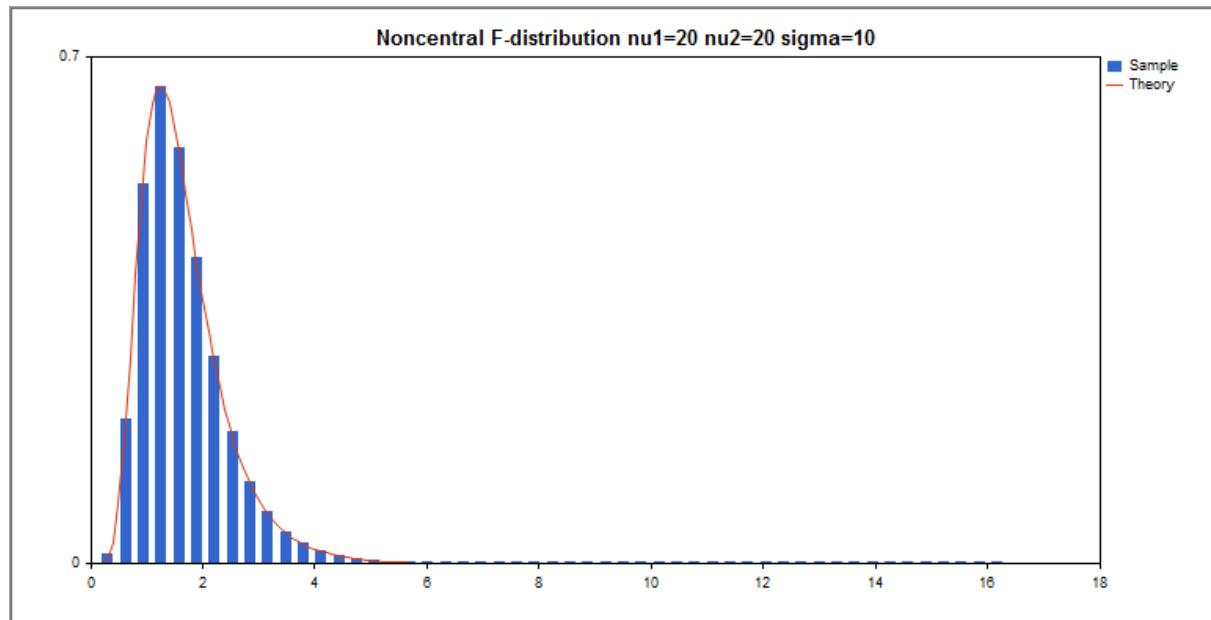
## Нецентральное F-распределение

В данном разделе представлены функции для работы с нецентральным F-распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по закону нецентрального распределения Фишера. Нецентральное F-распределение описывается следующей формулой:

$$f_{\text{NoncentralF}}(x | \nu_1, \nu_2, \sigma) = e^{-\frac{\sigma}{2}} \sum_{r=0}^{\infty} \frac{1}{r!} \left(\frac{\sigma}{2}\right)^r \frac{\Gamma\left(\frac{\nu_1 + \nu_2}{2} + r\right)}{\Gamma\left(\frac{\nu_2}{2} + r\right)\Gamma\left(\frac{\nu_1}{2}\right)} \left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_2}{2}+r} \frac{x^{\frac{\nu_2}{2}-1+r}}{\left(1 + \frac{\nu_1}{\nu_2}x\right)^{\frac{\nu_1+\nu_2}{2}+r}}$$

где:

- $x$  — значение случайной величины
- $\nu_1$  — первый параметр распределения (число степеней свободы)
- $\nu_2$  — второй параметр распределения (число степеней свободы)
- $\sigma$  — параметр нецентральности



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityNoncentralF</a>	Рассчитывает плотность вероятности нецентрального F-распределения
<a href="#">MathCumulativeDistributionNoncentralF</a>	Рассчитывает значение функции F-распределения вероятностей

<a href="#"><u>MathQuantileNoncentralF</u></a>	Рассчитывает значение обратной функции нецентрального F-распределения для заданной вероятности
<a href="#"><u>MathRandomNoncentralF</u></a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по закону нецентрального распределения Фишера
<a href="#"><u>MathMomentsNoncentralF</u></a>	Рассчитывает теоретические численные значения первых 4 моментов нецентрального распределения Фишера

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralF.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_1=20;           // первое число степеней свободы
input double nu_2=20;           // второе число степеней свободы
input double sig=10;            // параметр нецентральности
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;                // количество значений в выборке
    int ncells=51;                 // количество интервалов в гистограмме
    double x[];                   // центры интервалов гистограммы
    double y[];                   // количество значений из выборки, попавших в интервал
    double data[];                // выборка случайных значений
    double max,min;               // максимальное и минимальное значения в выборке
//--- получим выборку из нецентрального F-распределения
    MathRandomNoncentralF(nu_1,nu_2,sig,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
```

```

double x2[];
double y2[];
MathSequence(min,max,step,x2);
MathProbabilityDensityNoncentralF(x2,nu_1,nu_2,sig,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<nCells; i++)
    y[i]/=k;
//--- выводим графики
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Noncentral F-distribution nu1=%G nu2=%G sigma=%G"));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
}

```

```
for(int i=0; i<size; i++)
{
    int ind=int((data[i]-minv)/width);
    if(ind>=cells) ind=cells-1;
    frequency[ind]++;
}
return (true);
}

//+-----+
//| Calculates values for sequence generation |
//+-----+

void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityNoncentralF

Рассчитывает плотность вероятности нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNoncentralF(
    const double x,           // значение случайной величины
    const double nu1,          // первый параметр распределения (число степеней свободы)
    const double nu2,          // второй параметр распределения (число степеней свободы)
    const double sigma,        // параметр нецентральности
    const bool log_mode,       // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNoncentralF(
    const double x,           // значение случайной величины
    const double nu1,          // первый параметр распределения (число степеней свободы)
    const double nu2,          // второй параметр распределения (число степеней свободы)
    const double sigma,        // параметр нецентральности
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [df\(\)](#) в R.

```
bool MathProbabilityDensityNoncentralF(
    const double& x[],         // массив со значениями случайной величины
    const double nu1,            // первый параметр распределения (число степеней свободы)
    const double nu2,            // второй параметр распределения (число степеней свободы)
    const double sigma,          // параметр нецентральности
    const bool log_mode,         // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]            // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathProbabilityDensityNoncentralF(
    const double& x[],         // массив со значениями случайной величины
    const double nu1,            // первый параметр распределения (число степеней свободы)
    const double nu2,            // второй параметр распределения (число степеней свободы)
    const double sigma,          // параметр нецентральности
    double& result[]            // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*nu1*

[in] Первый параметр распределения (число степеней свободы).

*nu2*

[in] Второй параметр распределения (число степеней свободы).

*sigma*

[in] Параметр нецентральности.

*log\_mode*

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionNoncentralF

Рассчитывает распределение вероятностей по закону нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionNoncentralF(
    const double x, // значение случайной величины
    const double nu1, // первый параметр распределения (число степеней свободы)
    const double nu2, // второй параметр распределения (число степеней свободы)
    const double sigma, // параметр нецентральности
    const double tail, // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode, // флаг расчета логарифма значения, если log_mode=true
    int& error_code // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей по закону нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionNoncentralF(
    const double x, // значение случайной величины
    const double nu1, // первый параметр распределения (число степеней свободы)
    const double nu2, // второй параметр распределения (число степеней свободы)
    const double sigma, // параметр нецентральности
    int& error_code // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей по закону нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для массива случайных величин x[]. В случае ошибки возвращает `false`. Аналог [pf\(\)](#) в R.

```
bool MathCumulativeDistributionNoncentralF(
    const double& x[], // массив со значениями случайной величины
    const double nu1, // первый параметр распределения (число степеней свободы)
    const double nu2, // второй параметр распределения (число степеней свободы)
    const double sigma, // параметр нецентральности
    const double tail, // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode, // флаг расчета логарифма значения, если log_mode=true
    double& result[] // массив для значений функции вероятности
);
```

Рассчитывает распределение вероятностей по закону нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для массива случайных величин x[]. В случае ошибки возвращает `false`.

```
bool MathCumulativeDistributionNoncentralF(
    const double& x[], // массив со значениями случайной величины
    const double nu1, // первый параметр распределения (число степеней свободы)
    const double nu2, // второй параметр распределения (число степеней свободы)
    const double sigma, // параметр нецентральности
    double& result[] // массив для значений функции вероятности
);
```

```
) ;
```

## Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*nu1*

[in] Первый параметр распределения (число степеней свободы).

*nu2*

[in] Второй параметр распределения (число степеней свободы).

*sigma*

[in] Параметр нецентральности.

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета логарифма значения. Если log\_mode=true, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileF

Рассчитывает значение обратной функции нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileF(
    const double probability,           // значение вероятности появления случайной величины
    const double nu1,                  // первый параметр распределения (число степеней свободы)
    const double nu2,                  // второй параметр распределения (число степеней свободы)
    const double sigma,                // параметр нецентральности
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileF(
    const double probability,           // значение вероятности появления случайной величины
    const double nu1,                  // первый параметр распределения (число степеней свободы)
    const double nu2,                  // второй параметр распределения (число степеней свободы)
    const double sigma,                // параметр нецентральности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`. Аналог [qf\(\)](#) в R.

```
double MathQuantileF(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double nu1,                  // первый параметр распределения (число степеней свободы)
    const double nu2,                  // второй параметр распределения (число степеней свободы)
    const double sigma,                // параметр нецентральности
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`.

```
bool MathQuantileF(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double nu1,                  // первый параметр распределения (число степеней свободы)
    const double nu2,                  // второй параметр распределения (число степеней свободы)
    double& result[]                 // массив со значениями квантилей
);
```

## Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*nu1*

[in] Первый параметр распределения (число степеней свободы).

*nu2*

[in] Второй параметр распределения (число степеней свободы).

*sigma*

[in] Параметр нецентральности.

*tail*

[in] Флаг расчета, если `false`, то расчет ведется для вероятности  $1.0 - \text{probability}$ .

*log\_mode*

[in] Флаг расчета, если `log_mode=true`, то расчет ведется для вероятности  $\text{Exp}(\text{probability})$ .

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomNoncentralF

Генерирует псевдослучайную величину, распределенную по закону нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma. В случае ошибки возвращает [NaN](#).

```
double MathRandomNoncentralF(
    const double nu1,           // первый параметр распределения (число степеней свободы)
    const double nu2,           // второй параметр распределения (число степеней свободы)
    const double sigma,         // параметр нецентральности
    int& error_code            // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, по закону нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma. В случае ошибки возвращает false. Аналог [rf\(\)](#) в R.

```
bool MathRandomNoncentralF(
    const double nu1,           // первый параметр распределения (число степеней свободы)
    const double nu2,           // второй параметр распределения (число степеней свободы)
    const double sigma,         // параметр нецентральности
    const int data_count,       // количество необходимых данных
    double& result[]            // массив со значениями псевдослучайных величин
);
```

### Параметры

*nu1*

[in] Первый параметр распределения (число степеней свободы).

*nu2*

[in] Второй параметр распределения (число степеней свободы).

*sigma*

[in] Параметр нецентральности.

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsNoncentralF

Рассчитывает теоретические численные значения первых 4 моментов нецентрального F-распределения Фишера с параметрами nu1, nu2 и sigma.

```
double MathMomentsNoncentralF(
    const double nu1,           // первый параметр распределения (число степеней свободы)
    const double nu2,           // второй параметр распределения (число степеней свободы)
    const double sigma,         // параметр нецентральности
    double& mean,              // переменная для среднего значения
    double& variance,          // переменная для дисперсии
    double& skewness,           // переменная для коэффициента асимметрии
    double& kurtosis,           // переменная для коэффициента эксцесса
    int& error_code            // переменная для кода ошибки
);
```

### Параметры

*nu1*

[in] Первый параметр распределения (число степеней свободы).

*nu2*

[in] Второй параметр распределения (число степеней свободы).

*sigma*

[in] Параметр нецентральности.

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если расчет моментов произведен успешно, иначе false.

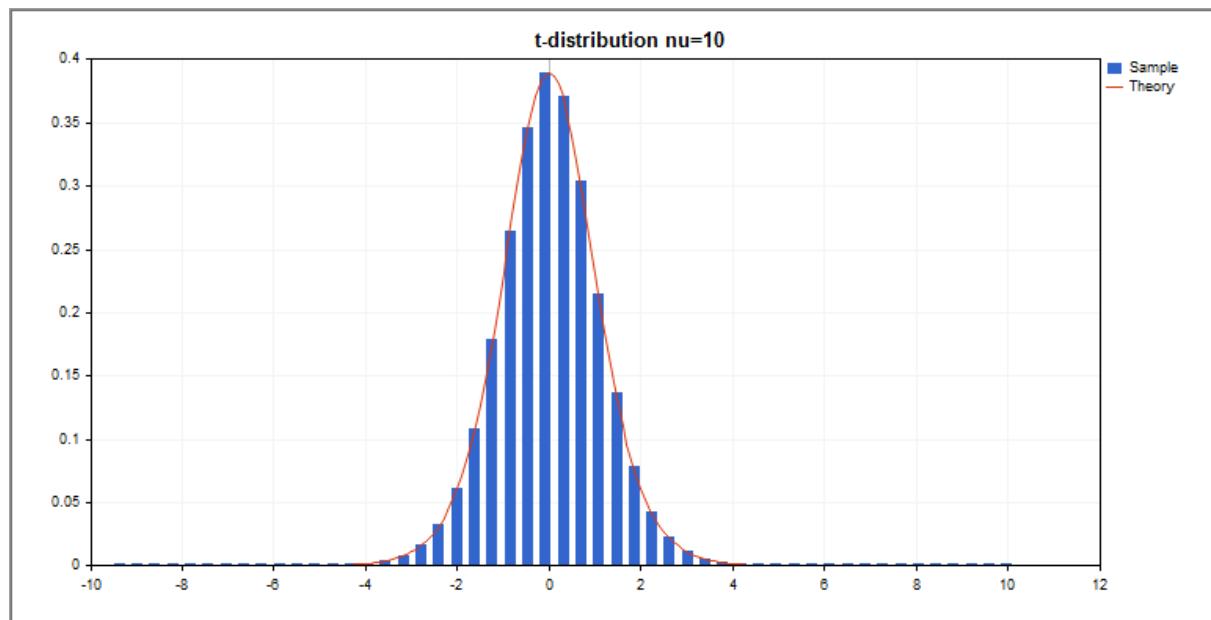
## Т-распределение

В данном разделе представлены функции для работы с Т-распределением Стьюдента. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по закону Стьюдента. Т-распределение описывается следующей формулой:

$$f_T(x|\nu) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)} \frac{1}{\sqrt{\pi\nu}} \frac{1}{\left(1+\frac{x^2}{\nu}\right)^{\frac{\nu+1}{2}}}$$

где:

- $x$  — значение случайной величины
- $\nu$  — параметр распределения (число степеней свободы)



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityT</a>	Рассчитывает плотность вероятности Т-распределения
<a href="#">MathCumulativeDistributionT</a>	Рассчитывает значение функции Т-распределения вероятностей
<a href="#">MathQuantileT</a>	Рассчитывает значение обратной функции Т-распределения для заданной вероятности
<a href="#">MathRandomT</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин,

	распределенных по закону Стьюдента
<a href="#">MathMomentsT</a>	Рассчитывает теоретические численные значения первых 4 моментов Т-распределения Стьюдента

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\T.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=10; // число степеней свободы
//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
long chart=0;
string name="GraphicNormal";
int n=1000000; // количество значений в выборке
int ncells=51; // количество интервалов в гистограмме
double x[]; // центры интервалов гистограммы
double y[]; // количество значений из выборки, попавших в интервал
double data[]; // выборка случайных значений
double max,min; // максимальное и минимальное значения в выборке
//--- получим выборку из Т-распределения
MathRandomT(nu_par,n,data);
//--- рассчитаем данные для построения гистограммы
CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
double step;
GetMaxMinStepValues(max,min,step);
step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
double x2[];
double y2[];
MathSequence(min,max,step,x2);
MathProbabilityDensityT(x2,nu_par,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
```

```

for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- ВЫВОДИМ ГРАФИКИ
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("t-distribution nu=%G",nu_par));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set | 
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```

```
//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityT

Рассчитывает плотность вероятности Т-распределения Стьюдента с параметром  $\nu$  для случайной величины  $x$ . В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityT(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается log(p)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности Т-распределения Стьюдента с параметром  $\nu$  для случайной величины  $x$ . В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityT(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности Т-распределения Стьюдента с параметром  $\nu$  для массива случайных величин  $x[]$ . В случае ошибки возвращает `false`. Аналог [dt\(\)](#) в R.

```
bool MathProbabilityDensityT(
    const double& x[],        // массив со значениями случайной величины
    const double nu,           // параметр распределения (число степеней свободы)
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается log(p)
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности Т-распределения Стьюдента с параметром  $\nu$  для массива случайных величин  $x[]$ . В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityT(
    const double& x[],        // массив со значениями случайной величины
    const double nu,           // параметр распределения (число степеней свободы)
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

$x$

[in] Значение случайной величины.

$x[]$

[in] Массив со значениями случайной величины.

$\nu$

[in] Параметр распределения (число степеней свободы).

$\log\_mode$

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

`error_code`

[out] Переменная для записи кода ошибки.

`result[]`

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionT

Рассчитывает распределение Стьюдента с параметром *nu* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionT(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const double tail,         // флаг расчета, если true, то рассчитывается вероятност
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает распределение Стьюдента с параметром *nu* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionT(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает распределение Стьюдента с параметром *nu* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [pt\(\)](#) в R.

```
bool MathCumulativeDistributionT(
    const double& x[],        // массив со значениями случайной величины
    const double nu,           // параметр распределения (число степеней свободы)
    const double tail,          // флаг расчета, если true, то рассчитывается вероятност
    const bool log_mode,        // флаг расчета логарифма значения, если log_mode=true
    double& result[]           // массив для значений функции вероятности
);
```

Рассчитывает распределение Стьюдента с параметром *nu* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathCumulativeDistributionT(
    const double& x[],        // массив со значениями случайной величины
    const double nu,           // параметр распределения (число степеней свободы)
    double& result[]           // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*nu*

[in] Параметр распределения (число степеней свободы).

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит x.

*log\_mode*

[in] Флаг расчета логарифма значения. Если log\_mode=true, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileT

Рассчитывает значение обратной функции Т-распределения Стьюдента с параметром *nu* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileT(
    const double probability,           // значение вероятности появления случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции Т-распределения Стьюдента с параметром *nu* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileT(
    const double probability,           // значение вероятности появления случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции Т-распределения Стьюдента с параметром *nu* для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`. Аналог [qt\(\)](#) в R.

```
double MathQuantileT(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double nu,                  // параметр распределения (число степеней свободы)
    const bool tail,                 // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    double& result[]                // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции Т-распределения Стьюдента с параметром *nu* для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`.

```
bool MathQuantileT(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double nu,                  // параметр распределения (число степеней свободы)
    double& result[]                // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*nu*

[in] Параметр распределения (число степеней свободы).

*tail*

[in] Флаг расчета, если false, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если log\_mode=true, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomT

Генерирует псевдослучайную величину, распределенную по закону Т-распределения Стьюдента с параметром nu. В случае ошибки возвращает [NaN](#).

```
double MathRandomT(
    const double nu,           // параметр распределения (число степеней свободы)
    int& error_code           // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону Т-распределения Стьюдента с параметром nu. В случае ошибки возвращает `false`. Аналог [rt\(\)](#) в R.

```
bool MathRandomT(
    const double nu,           // параметр распределения (число степеней свободы)
    const int data_count,      // количество необходимых данных
    double& result[]           // массив со значениями псевдослучайных величин
);
```

### Параметры

*nu*

[in] Параметр распределения (число степеней свободы).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsT

Рассчитывает теоретические численные значения первых 4 моментов Т-распределения Стьюдента с параметром *nu*.

```
double MathMomentsT(
    const double nu,           // параметр распределения (число степеней свободы)
    double& mean,             // переменная для среднего значения
    double& variance,         // переменная для дисперсии
    double& skewness,          // переменная для коэффициента асимметрии
    double& kurtosis,          // переменная для коэффициента эксцесса
    int& error_code           // переменная для кода ошибки
);
```

### Параметры

*nu*

[in] Параметр распределения (число степеней свободы).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает `true`, если расчет моментов произведен успешно, иначе `false`.

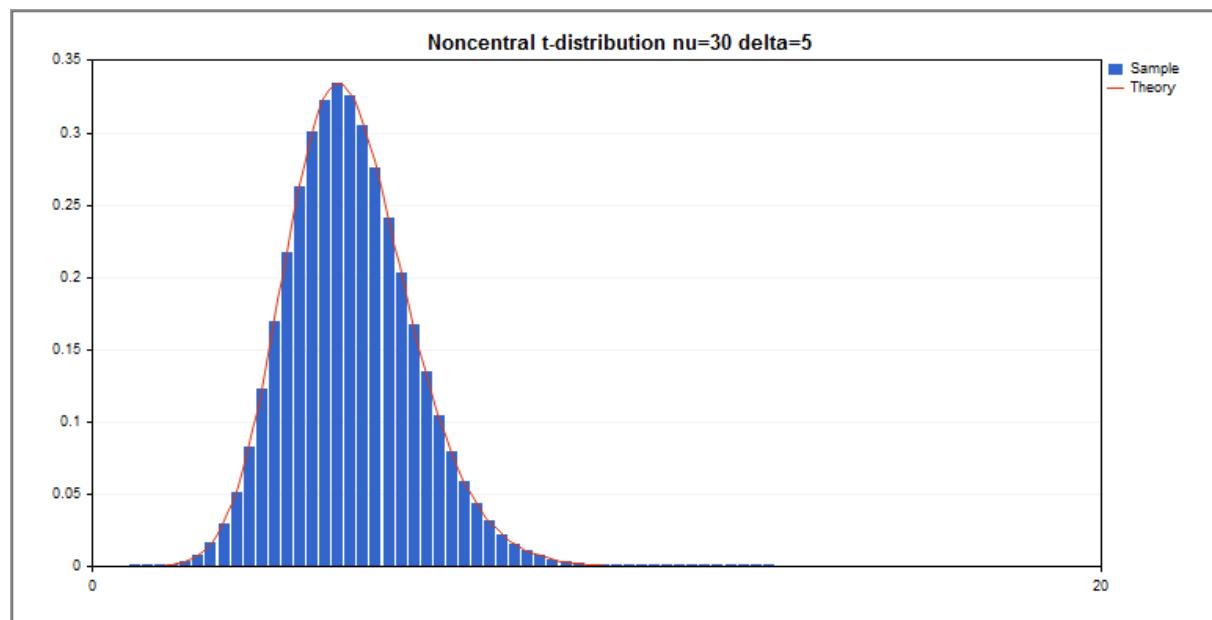
## T-распределение

В данном разделе представлены функции для работы с нецентральным Т-распределением Стьюдента. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по закону нецентрального Т-распределения. Нецентральное Т-распределение описывается следующей формулой:

$$f_{NoncentralT}(x | \nu, \delta) = \frac{\nu^{\frac{\nu}{2}} e^{-\frac{\delta^2}{2}}}{\Gamma\left(\frac{\nu}{2}\right) \sqrt{\pi} \left(\nu + x^2\right)^{\frac{\nu+1}{2}}} \sum_{r=0}^{\infty} \frac{(\nu\delta)^r}{r!} \left(\frac{2}{\nu + x^2}\right)^{\frac{r}{2}} \Gamma\left(\frac{\nu+r+1}{2}\right)$$

где:

- $x$  — значение случайной величины
- $\nu$  — параметр распределения (число степеней свободы)
- $\delta$  — параметр нецентральности



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityNoncentralT</a>	Рассчитывает плотность вероятности нецентрального Т-распределения
<a href="#">MathCumulativeDistributionNoncentralT</a>	Рассчитывает значение функции вероятностей нецентрального Т-распределения
<a href="#">MathQuantileNoncentralT</a>	Рассчитывает значение обратной функции нецентрального Т-распределения для заданной вероятности

<a href="#"><u>MathRandomNoncentralT</u></a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по закону нецентрального Т-распределения Стьюдента
<a href="#"><u>MathMomentsNoncentralT</u></a>	Рассчитывает теоретические численные значения первых 4 моментов нецентрального Т-распределения Стьюдента

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NoncentralT.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double nu_par=30;           // число степеней свободы
input double delta_par=5;          // параметр нецентральности
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;           // количество значений в выборке
    int ncells=51;            // количество интервалов в гистограмме
    double x[];               // центры интервалов гистограммы
    double y[];               // количество значений из выборки, попавших в интервал
    double data[];             // выборка случайных значений
    double max,min;            // максимальное и минимальное значения в выборке
//--- получим выборку из нецентрального Т-распределения
    MathRandomNoncentralT(nu_par,delta_par,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityNoncentralT(x2,nu_par,delta_par,false,y2);
```

```

//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- выводим графики
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Noncentral t-distribution nu=%G delta=%G",nu_,delta));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
    }
}

```

```
        frequency[ind]++;
    }
    return (true);
}
//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityNoncentralT

Рассчитывает плотность вероятности нецентрального Т-распределения Стьюдента с параметрами nu и delta для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNoncentralT(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const double delta,        // параметр нецентральности
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code           // переменная для кода ошибки
);
```

Рассчитывает плотность вероятности нецентрального Т-распределения Стьюдента с параметрами nu и delta для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNoncentralT(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const double delta,        // параметр нецентральности
    int& error_code           // переменная для кода ошибки
);
```

Рассчитывает плотность вероятности нецентрального Т-распределения Стьюдента с параметрами nu и delta для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [dt\(\)](#) в R.

```
bool MathProbabilityDensityNoncentralT(
    const double& x[],         // массив со значениями случайной величины
    const double nu,            // параметр распределения (число степеней свободы)
    const double delta,          // параметр нецентральности
    const bool log_mode,        // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]           // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности нецентрального Т-распределения Стьюдента с параметрами nu и delta для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathProbabilityDensityNoncentralT(
    const double& x[],         // массив со значениями случайной величины
    const double nu,            // параметр распределения (число степеней свободы)
    const double delta,          // параметр нецентральности
    double& result[]           // массив для значений функции плотности вероятности
);
```

### Параметры

x

[in] Значение случайной величины.

x[]

[in] Массив со значениями случайной величины.

*nu*

[in] Параметр распределения (число степеней свободы).

*delta*

[in] Параметр нецентральности.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionNoncentralT

Рассчитывает распределение вероятностей нецентрального Т-распределения с параметрами nu и delta для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionNoncentralT(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const double delta,        // параметр нецентральности
    const double tail,         // флаг расчета, если true, то рассчитывается вероятно-
                               // сть
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей нецентрального Т-распределения с параметрами nu и delta для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionNoncentralT(
    const double x,           // значение случайной величины
    const double nu,          // параметр распределения (число степеней свободы)
    const double delta,        // параметр нецентральности
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей нецентрального Т-распределения с параметрами nu и delta для массива случайных величин x[]. В случае ошибки возвращает [false](#). Аналог [pt\(\)](#) в R.

```
bool MathCumulativeDistributionNoncentralT(
    const double& x[],          // массив со значениями случайной величины
    const double nu,             // параметр распределения (число степеней свободы)
    const double delta,          // параметр нецентральности
    const double tail,           // флаг расчета, если true, то рассчитывается вероятно-
                               // сть
    const bool log_mode,         // флаг расчета логарифма значения, если log_mode=true
    double& result[]            // массив для значений функции вероятности
);
```

Рассчитывает распределение вероятностей нецентрального Т-распределения с параметрами nu и delta для массива случайных величин x[]. В случае ошибки возвращает [false](#).

```
bool MathCumulativeDistributionNoncentralT(
    const double& x[],          // массив со значениями случайной величины
    const double nu,             // параметр распределения (число степеней свободы)
    const double delta,          // параметр нецентральности
    double& result[]            // массив для значений функции вероятности
);
```

### Параметры

x

[in] Значение случайной величины.

x[]

[in] Массив со значениями случайной величины.

*nu*

[in] Параметр распределения (число степеней свободы).

*delta*

[in] Параметр нецентральности.

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит x.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode*=true, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileNoncentralT

Рассчитывает значение обратной функции нецентрального Т-распределения Стьюдента с параметрами nu и delta для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileNoncentralT(
    const double probability,           // значение вероятности появления случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const double delta,                // параметр нецентральности
    const bool tail,                  // флаг расчета, если lower_tail=false, то расчет ведется вправо от квантиля
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции нецентрального Т-распределения Стьюдента с параметрами nu и delta для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileNoncentralT(
    const double probability,           // значение вероятности появления случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const double delta,                // параметр нецентральности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции нецентрального Т-распределения Стьюдента с параметрами nu и delta для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*. Аналог [qt\(\)](#) в R.

```
double MathQuantileNoncentralT(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const double delta,                // параметр нецентральности
    const bool tail,                  // флаг расчета, если lower_tail=false, то расчет ведется вправо от квантиля
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции нецентрального Т-распределения Стьюдента с параметрами nu и delta для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*.

```
bool MathQuantileNoncentralT(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double nu,                   // параметр распределения (число степеней свободы)
    const double delta,                // параметр нецентральности
    double& result[]                 // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*nu*

[in] Параметр распределения (число степеней свободы).

*delta*

[in] Параметр нецентральности.

*tail*

[in] Флаг расчета, если *false*, то расчет ведется для вероятности 1.0-*probability*.

*log\_mode*

[in] Флаг расчета, если *log\_mode=true*, то расчет ведется для вероятности Exp(*probability*).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomNoncentralT

Генерирует псевдослучайную величину, распределенную по закону нецентрального Т-распределения Стьюдента с параметрами nu и delta. В случае ошибки возвращает [NaN](#).

```
double MathRandomNoncentralT(
    const double nu,           // параметр распределения (число степеней свободы)
    const double delta,         // параметр нецентральности
    int& error_code            // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону нецентрального Т-распределения Стьюдента с параметрами nu и delta. В случае ошибки возвращает false. Аналог [rt\(\)](#) в R.

```
bool MathRandomNoncentralT(
    const double nu,           // параметр распределения (число степеней свободы)
    const double delta,         // параметр нецентральности
    const int data_count,       // количество необходимых данных
    double& result[]           // массив со значениями псевдослучайных величин
);
```

### Параметры

*nu*

[in] Параметр распределения (число степеней свободы).

*delta*

[in] Параметр нецентральности.

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsNoncentralT

Рассчитывает теоретические численные значения первых 4 моментов нецентрального Т-распределения Стьюдента с параметрами nu и delta.

```
double MathMomentsNoncentralT(
    const double nu,           // параметр распределения (число степеней свободы)
    const double delta,         // параметр нецентральности
    double& mean,              // переменная для среднего значения
    double& variance,          // переменная для дисперсии
    double& skewness,           // переменная для коэффициента асимметрии
    double& kurtosis,           // переменная для коэффициента эксцесса
    int& error_code            // переменная для кода ошибки
);
```

### Параметры

*nu*

[in] Параметр распределения (число степеней свободы).

*delta*

[in] Параметр нецентральности.

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если расчет моментов произведен успешно, иначе false.

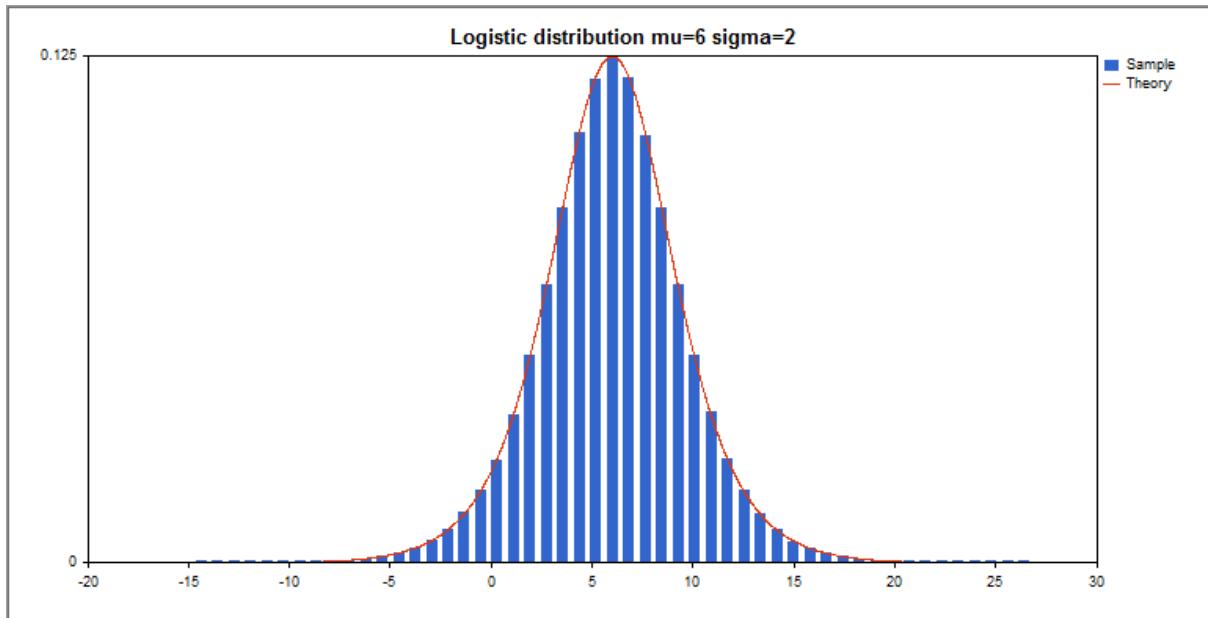
## Логистическое распределение

В данном разделе представлены функции для работы с логистическим распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по логистическому закону. Логистическое распределение описывается следующей формулой:

$$f_{Logistic}(x | \mu, \sigma) = \frac{e^{\frac{x-\mu}{\sigma}}}{\sigma \left(1 + e^{\frac{x-\mu}{\sigma}}\right)^2}$$

где:

- $x$  — значение случайной величины
- $\mu$  — параметр распределения mean
- $\sigma$  — параметр распределения scale



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityLogistic</a>	Рассчитывает плотность вероятности логистического распределения
<a href="#">MathCumulativeDistributionLogistic</a>	Рассчитывает значение функции логистического распределения вероятностей
<a href="#">MathQuantileLogistic</a>	Рассчитывает значение обратной функции логистического распределения для заданной вероятности

<a href="#">MathRandomLogistic</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по логистическому закону
<a href="#">MathMomentsLogistic</a>	Рассчитывает теоретические численные значения первых 4 моментов логистического распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Logistic.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double mu_par=6;           // параметр распределения mean
input double sigma_par=2;         // параметр распределения scale
//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;           // количество значений в выборке
    int ncells=51;           // количество интервалов в гистограмме
    double x[];              // центры интервалов гистограммы
    double y[];              // количество значений из выборки, попавших в интервал
    double data[];            // выборка случайных значений
    double max,min;          // максимальное и минимальное значения в выборке
//--- получим выборку из логистического распределения
    MathRandomLogistic(mu_par,sigma_par,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityLogistic(x2,mu_par,sigma_par,false,y2);
//--- масштабируем
```

```

double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- выводим графики
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Logistic distribution mu=%G sigma=%G",mu_par,sigma));
graphic.BackgroundMainSize(16);
//--- отключим автомасштабирование оси Y
graphic.YAxis().AutoScale(false);
graphic.YAxis().Max(theor_max);
graphic.YAxis().Min(0);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |+
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)

```

```
{  
    int ind=int((data[i]-minv)/width);  
    if(ind>=cells) ind=cells-1;  
    frequency[ind]++;  
}  
return (true);  
}  
//-----+  
//| Calculates values for sequence generation |  
//-----+  
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)  
{  
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации  
    double range=MathAbs(maxv-minv);  
    int degree=(int)MathRound(MathLog10(range));  
//--- нормализуем макс. и мин. значения с заданной точностью  
    maxv=NormalizeDouble(maxv,degree);  
    minv=NormalizeDouble(minv,degree);  
//--- шаг генерации последовательности также зададим от заданной точности  
    stepv=NormalizeDouble(MathPow(10,-degree),degree);  
    if((maxv-minv)/stepv<10)  
        stepv/=10.;  
}
```

## MathProbabilityDensityLogistic

Рассчитывает плотность вероятности логистического распределения с параметрами mu и sigma для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityLogistic(
    const double x,           // значение случайной величины
    const double mu,          // параметр распределения mean
    const double sigma,        // параметр распределения scale
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается ln(p)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности логистического распределения с параметрами mu и sigma для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityLogistic(
    const double x,           // значение случайной величины
    const double mu,          // параметр распределения mean
    const double sigma,        // параметр распределения scale
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности логистического распределения с параметрами mu и sigma для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [dlogis\(\)](#) в R.

```
bool MathProbabilityDensityLogistic(
    const double& x[],         // массив со значениями случайной величины
    const double mu,            // параметр распределения mean
    const double sigma,          // параметр распределения scale
    const bool log_mode,        // флаг расчета логарифма значения, если log_mode=true, то возвращается ln(p)
    double& result[]           // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности логистического распределения с параметрами mu и sigma для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathProbabilityDensityLogistic(
    const double& x[],         // массив со значениями случайной величины
    const double mu,            // параметр распределения mean
    const double sigma,          // параметр распределения scale
    double& result[]           // массив для значений функции плотности вероятности
);
```

### Параметры

x

[in] Значение случайной величины.

x[]

[in] Массив со значениями случайной величины.

*mu*

[in] Параметр распределения mean.

*sigma*

[in] Параметр распределения scale.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionLogistic

Рассчитывает логистическое распределение вероятностей с параметрами *mu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionLogistic(
    const double x,           // значение случайной величины
    const double mu,          // параметр распределения mean
    const double sigma,        // параметр распределения scale
    const double tail,         // флаг расчета, если true, то рассчитывается вероятно
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает логистическое распределение вероятностей с параметрами *mu* и *sigma* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionLogistic(
    const double x,           // значение случайной величины
    const double mu,          // параметр распределения mean
    const double sigma,        // параметр распределения scale
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает логистическое распределение вероятностей с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает *false*.

```
bool MathCumulativeDistributionLogistic(
    const double& x[],          // массив со значениями случайной величины
    const double mu,             // параметр распределения mean
    const double sigma,          // параметр распределения scale
    const double tail,           // флаг расчета, если true, то рассчитывается вероятно
    const bool log_mode,         // флаг расчета логарифма значения, если log_mode=true
    double& result[]            // массив для значений функции вероятности
);
```

Рассчитывает логистическое распределение вероятностей с параметрами *mu* и *sigma* для массива случайных величин *x[]*. В случае ошибки возвращает *false*. Аналог [plogis\(\)](#) в R.

```
bool MathCumulativeDistributionLogistic(
    const double& x[],          // массив со значениями случайной величины
    const double mu,             // параметр распределения mean
    const double sigma,          // параметр распределения scale
    double& result[]            // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*mu*

[in] Параметр распределения mean.

*sigma*

[in] Параметр распределения scale.

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит x.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode*=true, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileLogistic

Рассчитывает значение обратной функции логистического распределения с параметрами *mu* и *sigma* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileLogistic(
    const double probability,           // значение вероятности появления случайной величины
    const double mu,                   // параметр распределения mean
    const double sigma,                // параметр распределения scale
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции логистического распределения с параметрами *mu* и *sigma* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileLogistic(
    const double probability,           // значение вероятности появления случайной величины
    const double mu,                   // параметр распределения mean
    const double sigma,                // параметр распределения scale
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции логистического распределения с параметрами *mu* и *sigma* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*. Аналог [glogis\(\)](#) в R.

```
double MathQuantileLogistic(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double mu,                   // параметр распределения mean
    const double sigma,                // параметр распределения scale
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции логистического распределения с параметрами *mu* и *sigma* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*.

```
bool MathQuantileLogistic(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double mu,                   // параметр распределения mean
    const double sigma,                // параметр распределения scale
    double& result[]                 // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*mu*

[in] Параметр распределения mean.

*sigma*

[in] Параметр распределения scale.

*tail*

[in] Флаг расчета, если false, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если log\_mode=true, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomLogistic

Генерирует псевдослучайную величину, распределенную по закону логистического распределения с параметрами *mu* и *sigma*. В случае ошибки возвращает [NaN](#).

```
double MathRandomLogistic(
    const double mu,           // параметр распределения mean
    const double sigma,         // параметр распределения scale
    int& error_code            // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону логистического распределения с параметрами *mu* и *sigma*. В случае ошибки возвращает false. Аналог [rlogis\(\)](#) в R.

```
bool MathRandomLogistic(
    const double mu,           // параметр распределения mean
    const double sigma,         // параметр распределения scale
    const int data_count,       // количество необходимых данных
    double& result[]           // массив со значениями псевдослучайных величин
);
```

### Параметры

*mu*

[in] Параметр распределения mean.

*sigma*

[in] Параметр распределения scale.

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsLogistic

Рассчитывает теоретические численные значения первых 4 моментов логистического распределения с параметрами *mu* и *sigma*.

```
double MathMomentsLogistic(
    const double mu,           // параметр распределения mean
    const double sigma,        // параметр распределения scale
    double& mean,             // переменная для среднего значения
    double& variance,         // переменная для дисперсии
    double& skewness,          // переменная для коэффициента асимметрии
    double& kurtosis,          // переменная для коэффициента эксцесса
    int& error_code           // переменная для кода ошибки
);
```

### Параметры

*mu*

[in] Параметр распределения mean.

*sigma*

[in] Параметр распределения scale.

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если расчет моментов произведен успешно, иначе false.

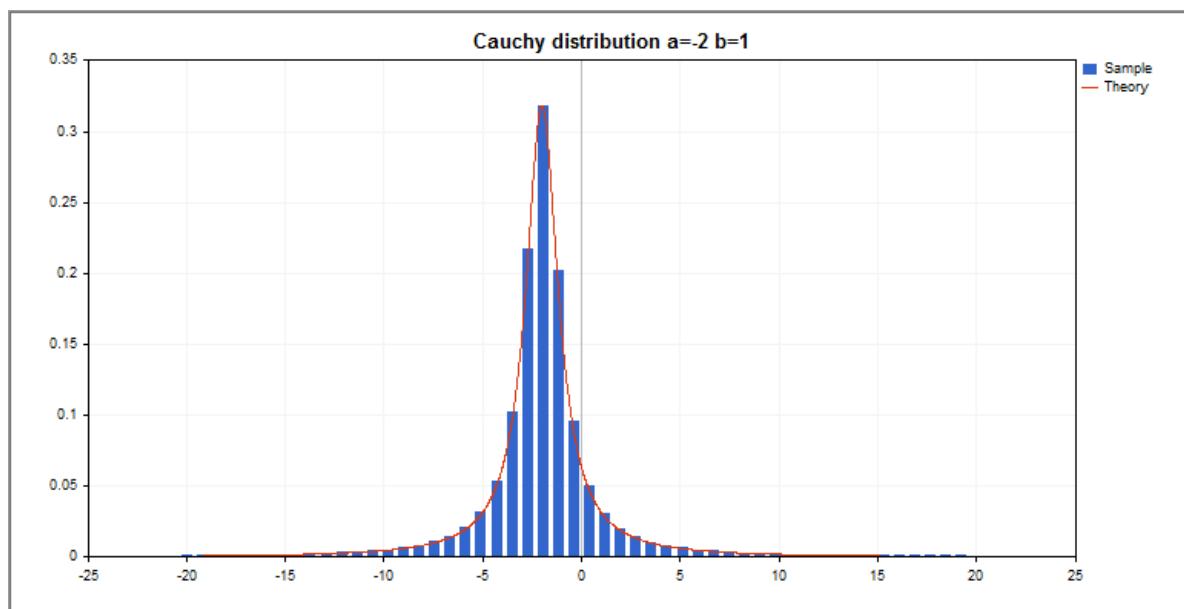
## Распределение Коши

В данном разделе представлены функции для работы с распределением Коши. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по закону Коши. Распределение Коши описывается следующей формулой:

$$f_{Cauchy}(x|a,b) = \frac{1}{\pi} \frac{b}{b^2 + (x-a)^2}$$

где:

- $x$  — значение случайной величины
- $a$  — параметр распределения mean
- $b$  — параметр распределения scale



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityCauchy</a>	Рассчитывает плотность вероятности распределения Коши
<a href="#">MathCumulativeDistributionCauchy</a>	Рассчитывает значение функции распределения вероятностей Коши
<a href="#">MathQuantileCauchy</a>	Рассчитывает значение обратной функции распределения Коши для заданной вероятности
<a href="#">MathRandomCauchy</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по закону Коши

MathMomentsCauchy

Рассчитывает теоретические численные значения первых 4 моментов распределения Коши

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Cauchy.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=-2;           // параметр распределения mean
input double b_par=1;             // параметр распределения scale
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;                 // количество значений в выборке
    int ncells=51;                  // количество интервалов в гистограмме
    double x[];                     // центры интервалов гистограммы
    double y[];                     // количество значений из выборки, попавших в интервал
    double data[];                  // выборка случайных значений
    double max,min;                // максимальное и минимальное значения в выборке
//--- получим выборку из распределения Коши
    MathRandomCauchy(a_par,b_par,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityCauchy(x2,a_par,b_par,false,y2);
//--- масштабируем
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
```

```

for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- выводим графики
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Cauchy distribution a=%G b=%G",a_par,b_par));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set | 
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1)
        return(false);
    int size=ArraySize(data);
    if(size<cells*10)
        return(false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    Print("min=",minv," max=",maxv);
    minv=-20;
    maxv=20;
    double range=maxv-minv;
    double width=range/cells;
    if(width==0)
        return(false);
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+i*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {

```

```
int ind=(int)MathRound((data[i]-minv)/width);
if(ind>=0 && ind<cells)
    frequency[ind]++;
}
return(true);
}

//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityCauchy

Рассчитывает плотность вероятности распределения Коши с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityCauchy(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения mean
    const double b,           // параметр распределения scale
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается ln(p)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности распределения Коши с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityCauchy(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения mean
    const double b,           // параметр распределения scale
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности распределения Коши с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог `dcauchy()` в R.

```
bool MathProbabilityDensityCauchy(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // параметр распределения mean
    const double b,            // параметр распределения scale
    const bool log_mode,      // флаг расчета логарифма значения, если log_mode=true, то возвращается ln(p)
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности распределения Коши с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityCauchy(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // параметр распределения mean
    const double b,            // параметр распределения scale
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*a*

[in] Параметр распределения mean.

*b*

[in] Параметр распределения scale.

*log\_mode*

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionCauchy

Рассчитывает распределение вероятностей Коши с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionCauchy(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения mean
    const double b,           // параметр распределения scale
    const double tail,        // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,      // флаг расчета логарифма значения, если log_mode=true
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей Коши с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionCauchy(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения mean
    const double b,           // параметр распределения scale
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает распределение вероятностей Коши с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает *false*.

```
bool MathCumulativeDistributionCauchy(
    const double& x[],         // массив со значениями случайной величины
    const double a,             // параметр распределения mean
    const double b,             // параметр распределения scale
    const double tail,          // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,        // флаг расчета логарифма значения, если log_mode=true
    double& result[]           // массив для значений функции вероятности
);
```

Рассчитывает распределение вероятностей Коши с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает *false*. Аналог [plogis\(\)](#) в R.

```
bool MathCumulativeDistributionCauchy(
    const double& x[],         // массив со значениями случайной величины
    const double a,             // параметр распределения mean
    const double b,             // параметр распределения scale
    double& result[]           // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*a*

[in] Параметр распределения mean.

*b*

[in] Параметр распределения scale.

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileCauchy

Рассчитывает значение обратной функции распределения Коши с параметрами *a* и *b* для вероятности *probability*. В случае ошибки возвращает *NaN*.

```
double MathQuantileCauchy(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // параметр распределения mean
    const double b,                     // параметр распределения scale
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифма вероятности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции распределения Коши с параметрами *a* и *b* для вероятности *probability*. В случае ошибки возвращает *NaN*.

```
double MathQuantileCauchy(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // параметр распределения mean
    const double b,                     // параметр распределения scale
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции распределения Коши с параметрами *a* и *b* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*. Аналог [qcauschy\(\)](#) в R.

```
double MathQuantileCauchy(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double a,                     // параметр распределения mean
    const double b,                     // параметр распределения scale
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифма вероятности
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции распределения Коши с параметрами *a* и *b* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*.

```
bool MathQuantileCauchy(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double a,                     // параметр распределения mean
    const double b,                     // параметр распределения scale
    double& result[]                 // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*a*

[in] Параметр распределения mean.

*b*

[in] Параметр распределения scale.

*tail*

[in] Флаг расчета, если false, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если log\_mode=true, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomCauchy

Генерирует псевдослучайную величину, распределенную по закону распределения Коши с параметрами *a* и *b*. В случае ошибки возвращает [NaN](#).

```
double MathRandomCauchy(
    const double a,           // параметр распределения mean
    const double b,           // параметр распределения scale
    int& error_code          // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону распределения Коши с параметрами *a* и *b*. В случае ошибки возвращает `false`. Аналог [rcauchy\(\)](#) в R.

```
bool MathRandomCauchy(
    const double a,           // параметр распределения mean
    const double b,           // параметр распределения scale
    const int data_count,     // количество необходимых данных
    double& result[]          // массив со значениями псевдослучайных величин
);
```

### Параметры

*a*

[in] Параметр распределения mean.

*b*

[in] Параметр распределения scale.

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsCauchy

Рассчитывает теоретические численные значения первых 4 моментов распределения Коши с параметрами *a* и *b*.

```
double MathMomentsCauchy(
    const double a,           // параметр распределения mean
    const double b,           // параметр распределения scale
    double& mean,            // переменная для среднего значения
    double& variance,        // переменная для дисперсии
    double& skewness,         // переменная для коэффициента асимметрии
    double& kurtosis,         // переменная для коэффициента эксцесса
    int& error_code          // переменная для кода ошибки
);
```

### Параметры

*a*

[in] Параметр распределения mean.

*b*

[in] Параметр распределения scale.

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если расчет моментов произведен успешно, иначе false.

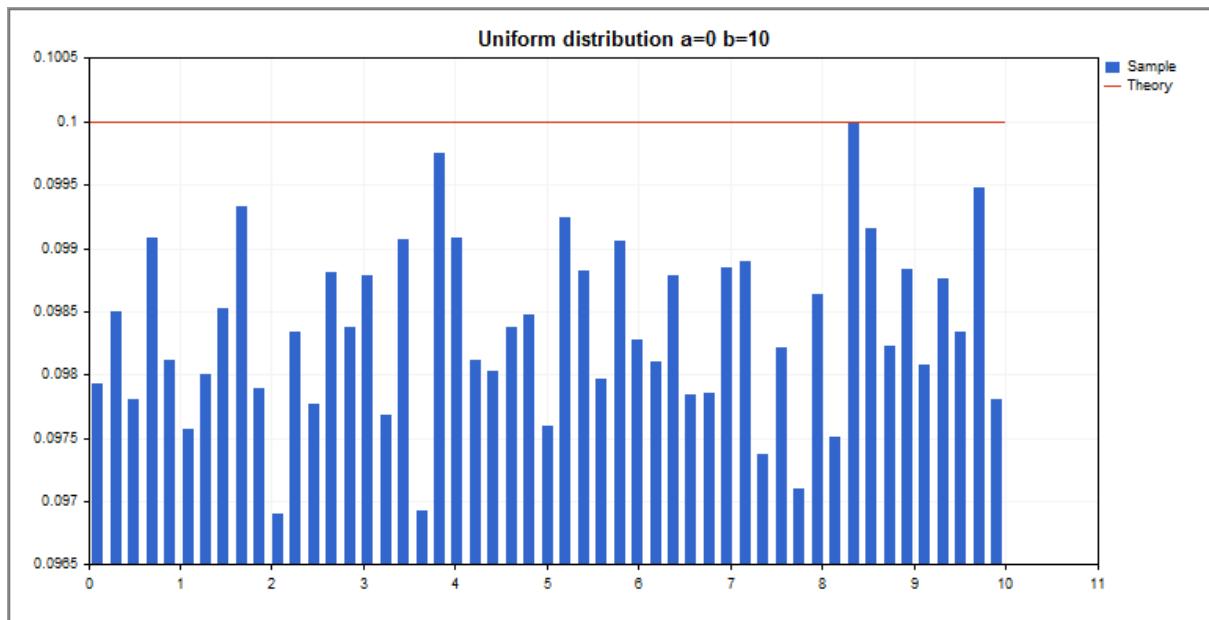
## Равномерное распределение

В данном разделе представлены функции для работы с равномерным распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по равномерному закону. Равномерное распределение описывается следующей формулой:

$$f_{Uniform}(x| a, b) = \frac{1}{b-a}$$

где:

- $x$  – значение случайной величины
- $a$  – параметр распределения (нижняя граница)
- $b$  – параметр распределения (верхняя граница)



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityUniform</a>	Рассчитывает плотность вероятности равномерного распределения
<a href="#">MathCumulativeDistributionUniform</a>	Рассчитывает значение функции равномерного распределения вероятностей
<a href="#">MathQuantileUniform</a>	Рассчитывает значение обратной функции равномерного распределения для заданной вероятности
<a href="#">MathRandomUniform</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по равномерному закону

MathMomentsUniform

Рассчитывает теоретические численные значения первых 4 моментов равномерного распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Uniform.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=0;           // параметр распределения а (нижняя граница)
input double b_par=10;          // параметр распределения b (верхняя граница)
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;             // количество значений в выборке
    int ncells=51;              // количество интервалов в гистограмме
    double x[];                 // центры интервалов гистограммы
    double y[];                 // количество значений из выборки, попавших в интервал
    double data[];              // выборка случайных значений
    double max,min;             // максимальное и минимальное значения в выборке
//--- получим выборку из равномерного распределения
    MathRandomUniform(a_par,b_par,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityUniform(x2,a_par,b_par,false,y2);
//--- масштабируем
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
```

```

for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- ВЫВОДИМ ГРАФИКИ
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Uniform distribution a=%G b=%G",a_par,b_par));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set | 
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```

```
//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityUniform

Рассчитывает плотность вероятности равномерного распределения с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityUniform(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения a (нижняя граница)
    const double b,           // параметр распределения b (верхняя граница)
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается ln(p(x))
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности равномерного распределения с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityUniform(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения a (нижняя граница)
    const double b,           // параметр распределения b (верхняя граница)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности равномерного распределения с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [dunif\(\)](#) в R.

```
bool MathProbabilityDensityUniform(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // параметр распределения a (нижняя граница)
    const double b,            // параметр распределения b (верхняя граница)
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается ln(p(x))
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности равномерного распределения с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityUniform(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // параметр распределения a (нижняя граница)
    const double b,            // параметр распределения b (верхняя граница)
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*a*

[in] Параметр распределения a (нижняя граница).

*b*

[in] Параметр распределения b (верхняя граница).

*log\_mode*

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionUniform

Рассчитывает равномерное распределение вероятностей с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionUniform(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения a (нижняя граница)
    const double b,           // параметр распределения b (верхняя граница)
    const double tail,        // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,      // флаг расчета логарифма значения, если log_mode=true
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает равномерное распределение вероятностей с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionUniform(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения a (нижняя граница)
    const double b,           // параметр распределения b (верхняя граница)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает равномерное распределение вероятностей с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает *false*.

```
bool MathCumulativeDistributionUniform(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // параметр распределения a (нижняя граница)
    const double b,            // параметр распределения b (верхняя граница)
    const double tail,         // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true
    double& result[]          // массив для значений функции вероятности
);
```

Рассчитывает равномерное распределение вероятностей с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает *false*. Аналог [punif\(\)](#) в R.

```
bool MathCumulativeDistributionUniform(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // параметр распределения a (нижняя граница)
    const double b,            // параметр распределения b (верхняя граница)
    double& result[]          // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*a*

[in] Параметр распределения а (нижняя граница).

*b*

[in] Параметр распределения b (верхняя граница).

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит x.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode*=true, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileUniform

Рассчитывает значение обратной функции равномерного распределения с параметрами *a* и *b* для вероятности *probability*. В случае ошибки возвращает *NaN*.

```
double MathQuantileUniform(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // параметр распределения a (нижняя граница)
    const double b,                     // параметр распределения b (верхняя граница)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифмической вероятности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции равномерного распределения с параметрами *a* и *b* для вероятности *probability*. В случае ошибки возвращает *NaN*.

```
double MathQuantileUniform(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // параметр распределения a (нижняя граница)
    const double b,                     // параметр распределения b (верхняя граница)
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции равномерного распределения с параметрами *a* и *b* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*. Аналог [qcauschy\(\)](#) в R.

```
double MathQuantileUniform(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double a,                     // параметр распределения a (нижняя граница)
    const double b,                     // параметр распределения b (верхняя граница)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифмической вероятности
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции равномерного распределения с параметрами *a* и *b* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*.

```
bool MathQuantileUniform(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double a,                     // параметр распределения a (нижняя граница)
    const double b,                     // параметр распределения b (верхняя граница)
    double& result[]                 // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*a*

[in] Параметр распределения а (нижняя граница).

*b*

[in] Параметр распределения b (верхняя граница).

*tail*

[in] Флаг расчета, если false, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если log\_mode=true, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomUniform

Генерирует псевдослучайную величину, распределенную по закону равномерного распределения с параметрами *a* и *b*. В случае ошибки возвращает `NaN`.

```
double MathRandomUniform(
    const double a,           // параметр распределения а (нижняя граница)
    const double b,           // параметр распределения b (верхняя граница)
    int&      error_code     // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону равномерного распределения с параметрами *a* и *b*. В случае ошибки возвращает `false`. Аналог `runif()` в R.

```
bool MathRandomUniform(
    const double a,           // параметр распределения а (нижняя граница)
    const double b,           // параметр распределения b (верхняя граница)
    const int   data_count,   // количество необходимых данных
    double&    result[]       // массив со значениями псевдослучайных величин
);
```

### Параметры

*a*

[in] Параметр распределения а (нижняя граница).

*b*

[in] Параметр распределения b (верхняя граница).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsUniform

Рассчитывает теоретические численные значения первых 4 моментов равномерного распределения с параметрами а и b.

```
double MathMomentsUniform(
    const double a,           // параметр распределения а (нижняя граница)
    const double b,           // параметр распределения b (верхняя граница)
    double& mean,            // переменная для среднего значения
    double& variance,        // переменная для дисперсии
    double& skewness,         // переменная для коэффициента асимметрии
    double& kurtosis,         // переменная для коэффициента эксцесса
    int& error_code          // переменная для кода ошибки
);
```

### Параметры

*a*

[in] Параметр распределения а (нижняя граница).

*b*

[in] Параметр распределения b (верхняя граница).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если расчет моментов произведен успешно, иначе false.

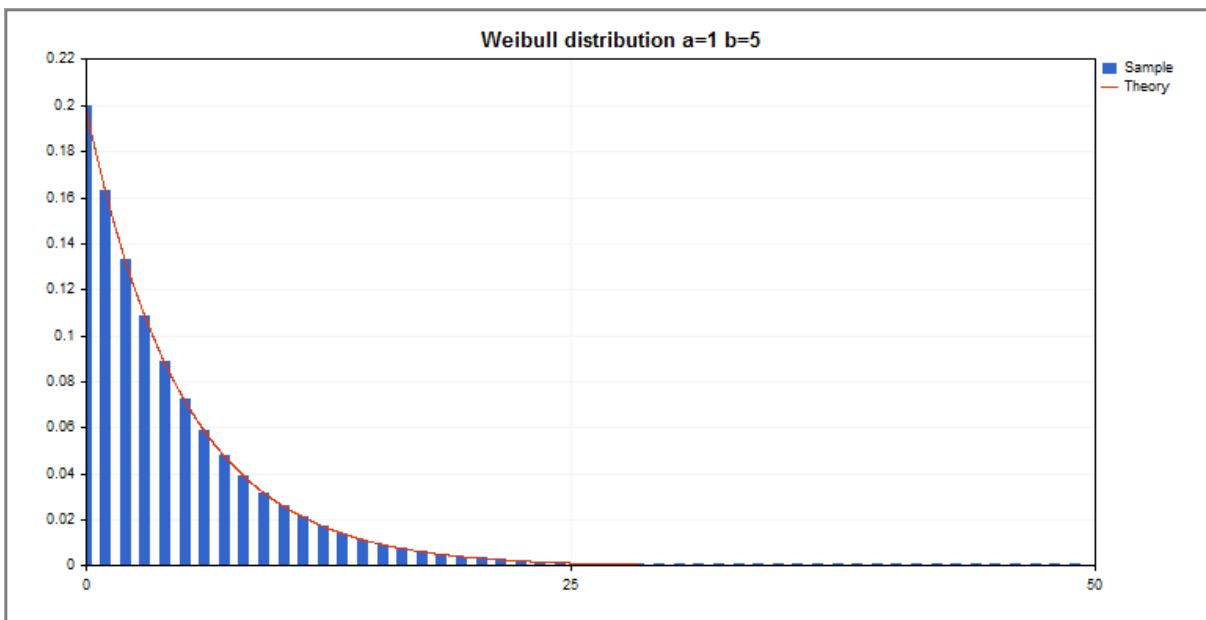
## Распределение Вейбулла

В данном разделе представлены функции для работы с распределением Вейбулла. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по закону Вейбулла. Распределение Вейбулла описывается следующей формулой:

$$f_{\text{Weibull}}(x|a,b) = \frac{a}{b} \left(\frac{x}{b}\right)^{a-1} e^{-\left(\frac{x}{b}\right)^a}$$

Где:

- $x$  – значение случайной величины
- $a$  – параметр распределения (shape)
- $b$  – параметр распределения (scale)



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityWeibull</a>	Рассчитывает плотность вероятности распределения Вейбулла
<a href="#">MathCumulativeDistributionWeibull</a>	Рассчитывает значение функции распределения вероятностей Вейбулла
<a href="#">MathQuantileWeibull</a>	Рассчитывает значение обратной функции распределения Вейбулла для заданной вероятности
<a href="#">MathRandomWeibull</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по закону Вейбулла

MathMomentsWeibull

Рассчитывает теоретические численные значения первых 4 моментов распределения Вейбулла

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Weibull.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double a_par=1;           // параметр распределения (shape)
input double b_par=5;           // параметр распределения (scale)
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;           // количество значений в выборке
    int ncells=51;           // количество интервалов в гистограмме
    double x[];              // центры интервалов гистограммы
    double y[];              // количество значений из выборки, попавших в интервал
    double data[];            // выборка случайных значений
    double max,min;          // максимальное и минимальное значения в выборке
//--- получим выборку из распределения Вейбулла
    MathRandomWeibull(a_par,b_par,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
    double step;
    GetMaxMinStepValues(max,min,step);
    step=MathMin(step,(max-min)/ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
    double x2[];
    double y2[];
    MathSequence(min,max,step,x2);
    MathProbabilityDensityWeibull(x2,a_par,b_par,false,y2);
//--- масштабируем
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
```

```

    for(int i=0; i<ncells; i++)
        y[i]/=k;
    //--- выводим графики
    CGraphic graphic;
    if(ObjectFind(chart,name)<0)
        graphic.Create(chart,name,0,0,0,780,380);
    else
        graphic.Attach(chart,name);
    graphic.BackgroundMain(StringFormat("Weibull distribution a=%G b=%G",a_par,b_par));
    graphic.BackgroundMainSize(16);
    //--- отключим автомасштабирование оси X
    graphic.XAxis().AutoScale(false);
    graphic.XAxis().Max(max);
    graphic.XAxis().Min(min);
    //--- plot all curves
    graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
    //--- а теперь построим теоретическую кривую плотности распределения
    graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
    graphic.CurvePlotAll();
    //--- plot all curves
    graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
    //--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+i*width;
        frequency[i]=0;
    }
    //--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
    }
}

```

```
        frequency[ind]++;
    }
    return (true);
}
//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityWeibull

Рассчитывает плотность вероятности распределения Вейбулла с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityWeibull(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения (shape)
    const double b,           // параметр распределения (scale)
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается ln(p)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности распределения Вейбулла с параметрами *a* и *b* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityWeibull(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения (shape)
    const double b,           // параметр распределения (scale)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает плотность вероятности распределения Вейбулла с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [dweibull\(\)](#) в R.

```
bool MathProbabilityDensityWeibull(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // параметр распределения (shape)
    const double b,            // параметр распределения (scale)
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается ln(p)
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает плотность вероятности распределения Вейбулла с параметрами *a* и *b* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityWeibull(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // параметр распределения (shape)
    const double b,            // параметр распределения (scale)
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*a*

[in] Параметр распределения (scale).

*b*

[in] Параметр распределения (shape).

*log\_mode*

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionWeibull

Рассчитывает распределение Вейбулла с параметрами a и b для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionWeibull(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения (shape)
    const double b,           // параметр распределения (scale)
    const double tail,        // флаг расчета, если true, то рассчитывается вероятно-
    const bool log_mode,      // флаг расчета логарифма значения, если log_mode=true
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает распределение Вейбулла с параметрами a и b для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionWeibull(
    const double x,           // значение случайной величины
    const double a,           // параметр распределения (shape)
    const double b,           // параметр распределения (scale)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает распределение Вейбулла с параметрами a и b для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [pweibull\(\)](#) в R.

```
bool MathCumulativeDistributionWeibull(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // параметр распределения (shape)
    const double b,            // параметр распределения (scale)
    const double tail,         // флаг расчета, если true, то рассчитывается вероятно-
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true
    double& result[]          // массив для значений функции вероятности
);
```

Рассчитывает распределение Вейбулла с параметрами a и b для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathCumulativeDistributionWeibull(
    const double& x[],        // массив со значениями случайной величины
    const double a,            // параметр распределения (shape)
    const double b,            // параметр распределения (scale)
    double& result[]          // массив для значений функции вероятности
);
```

### Параметры

x

[in] Значение случайной величины.

x[]

[in] Массив со значениями случайной величины.

*a*

[in] Параметр распределения (scale).

*b*

[in] Параметр распределения (shape).

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileWeibull

Рассчитывает значение обратной функции распределения вероятностей Вейбулла с параметрами *a* и *b* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileWeibull(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // параметр распределения (shape)
    const double b,                     // параметр распределения (scale)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифма вероятности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции распределения вероятностей Вейбулла с параметрами *a* и *b* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileWeibull(
    const double probability,           // значение вероятности появления случайной величины
    const double a,                     // параметр распределения (shape)
    const double b,                     // параметр распределения (scale)
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает значение обратной функции распределения вероятностей Вейбулла с параметрами *a* и *b* для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`. Аналог [qweibull\(\)](#) в R.

```
double MathQuantileWeibull(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double a,                     // параметр распределения (shape)
    const double b,                     // параметр распределения (scale)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифма вероятности
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает значение обратной функции распределения вероятностей Вейбулла с параметрами *a* и *b* для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`.

```
bool MathQuantileWeibull(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double a,                     // параметр распределения (shape)
    const double b,                     // параметр распределения (scale)
    double& result[]                 // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*a*

[in] Параметр распределения (scale).

*b*

[in] Параметр распределения (shape).

*tail*

[in] Флаг расчета, если *false*, то расчет ведется для вероятности 1.0-*probability*.

*log\_mode*

[in] Флаг расчета, если *log\_mode=true*, то расчет ведется для вероятности Exp(*probability*).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomWeibull

Генерирует псевдослучайную величину, распределенную по закону распределения Вейбулла с параметрами *a* и *b*. В случае ошибки возвращает [NaN](#).

```
double MathRandomWeibull(
    const double a,           // параметр распределения (shape)
    const double b,           // параметр распределения (scale)
    int& error_code          // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону распределения Вейбулла с параметрами *a* и *b*. В случае ошибки возвращает `false`. Аналог [rweibull\(\)](#) в R.

```
bool MathRandomWeibull(
    const double a,           // параметр распределения (shape)
    const double b,           // параметр распределения (scale)
    const int data_count,     // количество необходимых данных
    double& result[]          // массив со значениями псевдослучайных величин
);
```

### Параметры

*a*

[in] Параметр распределения (scale).

*b*

[in] Параметр распределения (shape).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsWeibull

Рассчитывает теоретические численные значения первых 4 моментов распределения Вейбулла с параметрами *a* и *b*.

```
double MathMomentsWeibull(
    const double a,           // параметр распределения (shape)
    const double b,           // параметр распределения (scale)
    double& mean,            // переменная для среднего значения
    double& variance,        // переменная для дисперсии
    double& skewness,         // переменная для коэффициента асимметрии
    double& kurtosis,         // переменная для коэффициента эксцесса
    int& error_code          // переменная для кода ошибки
);
```

### Параметры

*a*

[in] Параметр распределения (scale).

*b*

[in] Параметр распределения (shape).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если расчет моментов произведен успешно, иначе false.

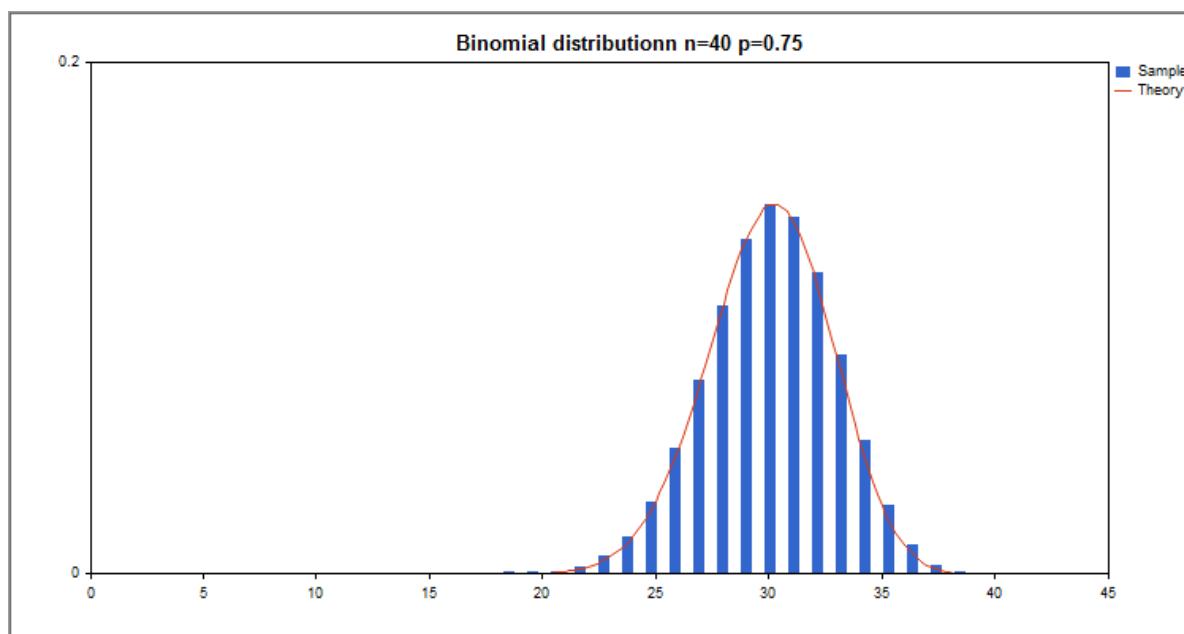
## Биномиальное распределение

В данном разделе представлены функции для работы с биномиальным распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по биномиальному закону. Биномиальное распределение описывается следующей формулой:

$$f_{Binomial}(x|n,p) = \binom{n}{x} p^x (1-p)^{n-x}$$

где:

- $x$  – значение случайной величины
- $n$  – количество испытаний
- $p$  – вероятность успеха для каждого испытания



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityBinomial</a>	Рассчитывает плотность вероятности биномиального распределения
<a href="#">MathCumulativeDistributionBinomial</a>	Рассчитывает значение функции биномиального распределения вероятностей
<a href="#">MathQuantileBinomial</a>	Рассчитывает значение обратной функции биномиального распределения для заданной вероятности
<a href="#">MathRandomBinomial</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по биномиальному закону

MathMomentsBinomial

Рассчитывает теоретические численные значения первых 4 моментов биномиального распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Binomial.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double n_par=40;           // количество испытаний
input double p_par=0.75;          // вероятность успеха для каждого испытания
//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;           // количество значений в выборке
    int ncells=20;           // количество интервалов в гистограмме
    double x[];              // центры интервалов гистограммы
    double y[];              // количество значений из выборки, попавших в интервал
    double data[];            // выборка случайных значений
    double max,min;          // максимальное и минимальное значения в выборке
//--- получим выборку из биномиального распределения
    MathRandomBinomial(n_par,p_par,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
    double x2[];
    double y2[];
    MathSequence(0,n_par,1,x2);
    MathProbabilityDensityBinomial(x2,n_par,p_par,false,y2);
//--- масштабируем
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
        y[i]/=k;
//--- выводим графики
    CGraphic graphic;
```

```

if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Binomial distributionn n=%G p=%G",n_par,p_par));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                            double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```

## MathProbabilityDensityBinomial

Рассчитывает значение функции вероятности (probability mass function) биномиального распределения с параметрами *n* и *p* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityBinomial(
    const double x,           // значение случайной величины
    const double n,           // параметр распределения (количество испытаний)
    const double p,           // параметр распределения (вероятность успеха для каждого испытания)
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает значение функции вероятности (probability mass function) биномиального распределения с параметрами *n* и *p* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityBinomial(
    const double x,           // значение случайной величины
    const double n,           // параметр распределения (количество испытаний)
    const double p,           // параметр распределения (вероятность успеха для каждого испытания)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает значение функции вероятности (probability mass function) биномиального распределения с параметрами *n* и *p* для массива случайных величин *x[]*. В случае ошибки возвращает *false*. Аналог [dbinom\(\)](#) в R.

```
bool MathProbabilityDensityBinomial(
    const double& x[],        // массив со значениями случайной величины
    const double n,            // параметр распределения (количество испытаний)
    const double p,            // параметр распределения (вероятность успеха для каждого испытания)
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает значение функции вероятности (probability mass function) биномиального распределения с параметрами *n* и *p* для массива случайных величин *x[]*. В случае ошибки возвращает *false*.

```
bool MathProbabilityDensityBinomial(
    const double& x[],        // массив со значениями случайной величины
    const double n,            // параметр распределения (количество испытаний)
    const double p,            // параметр распределения (вероятность успеха для каждого испытания)
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*n*

[in] Параметр распределения (количество испытаний).

*p*

[in] Параметр распределения (вероятность успеха для каждого испытания).

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionBinomial

Рассчитывает функцию распределения для биномиального закона с параметрами *n* и *p* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionBinomial(
    const double x,           // значение случайной величины
    const double n,           // параметр распределения (количество испытаний)
    const double p,           // параметр распределения (вероятность успеха для каждого испытания)
    const bool tail,          // флаг расчета, если true, то рассчитывается вероятность х и меньше, иначе - больше
    const bool log_mode,      // флаг расчета логарифма значения, если log_mode=true
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает функцию распределения для биномиального закона с параметрами *n* и *p* для массива случайных величин *x*[*n*]. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionBinomial(
    const double x[],          // значение случайной величины
    const double n,           // параметр распределения (количество испытаний)
    const double p,           // параметр распределения (вероятность успеха для каждого испытания)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает функцию распределения для биномиального закона с параметрами *n* и *p* для массива случайных величин *x*[*n*]. В случае ошибки возвращает [false](#). Аналог [pweibull\(\)](#) в R.

```
bool MathCumulativeDistributionBinomial(
    const double& x[],          // массив со значениями случайной величины
    const double n,           // параметр распределения (количество испытаний)
    const double p,           // параметр распределения (вероятность успеха для каждого испытания)
    const bool tail,          // флаг расчета, если true, то рассчитывается вероятность х и меньше, иначе - больше
    const bool log_mode,      // флаг расчета логарифма значения, если log_mode=true
    double& result[]          // массив для значений функции вероятности
);
```

Рассчитывает функцию распределения для биномиального закона с параметрами *n* и *p* для массива случайных величин *x*[*n*]. В случае ошибки возвращает [false](#).

```
bool MathCumulativeDistributionBinomial(
    const double& x[],          // массив со значениями случайной величины
    const double n,           // параметр распределения (количество испытаний)
    const double p,           // параметр распределения (вероятность успеха для каждого испытания)
    double& result[]          // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x*[]

[in] Массив со значениями случайной величины.

*n*

[in] Параметр распределения (количество испытаний).

*P*

[in] Параметр распределения (вероятность успеха для каждого испытания).

*tail*

[in] Флаг расчета. Если true, то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileBinomial

Рассчитывает обратное значение функции распределения для биномиального закона с параметрами *n* и *p* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileBinomial(
    const double probability,           // значение вероятности появления случайной величины
    const double n,                     // параметр распределения (количество испытаний)
    const double p,                     // параметр распределения (вероятность успеха для каждого испытания)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифма от вероятности
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает обратное значение функции распределения для биномиального закона с параметрами *n* и *p* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileBinomial(
    const double probability,           // значение вероятности появления случайной величины
    const double n,                     // параметр распределения (количество испытаний)
    const double p,                     // параметр распределения (вероятность успеха для каждого испытания)
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает обратное значение функции распределения для биномиального закона с параметрами *n* и *p* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*. Аналог [qbinom\(\)](#) в R.

```
double MathQuantileBinomial(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double n,                     // параметр распределения (количество испытаний)
    const double p,                     // параметр распределения (вероятность успеха для каждого испытания)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифма от вероятности
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает обратное значение функции распределения для биномиального закона с параметрами *n* и *p* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*.

```
bool MathQuantileBinomial(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double n,                     // параметр распределения (количество испытаний)
    const double p,                     // параметр распределения (вероятность успеха для каждого испытания)
    double& result[]                 // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*n*

[in] Параметр распределения (количество испытаний).

*p*

[in] Параметр распределения (вероятность успеха для каждого испытания).

*tail*

[in] Флаг расчета, если *false*, то расчет ведется для вероятности 1.0-*probability*.

*log\_mode*

[in] Флаг расчета, если *log\_mode=true*, то расчет ведется для вероятности  $\text{Exp}(\text{probability})$ .

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomBinomial

Генерирует псевдослучайную величину, распределенную по закону биномиального распределения с параметрами *n* и *p*. В случае ошибки возвращает `NaN`.

```
double MathRandomBinomial(
    const double n,           // параметр распределения (количество испытаний)
    const double p,           // параметр распределения (вероятность успеха для каждого испытания)
    int& error_code          // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону биномиального распределения с параметрами *n* и *p*. В случае ошибки возвращает `false`. Аналог `rweibull()` в R.

```
bool MathRandomBinomial(
    const double n,           // параметр распределения (количество испытаний)
    const double p,           // параметр распределения (вероятность успеха для каждого испытания)
    const int data_count,     // количество необходимых данных
    double& result[]          // массив со значениями псевдослучайных величин
);
```

### Параметры

*n*

[in] Параметр распределения (количество испытаний).

*p*

[in] Параметр распределения (вероятность успеха для каждого испытания).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsBinomial

Рассчитывает теоретические численные значения первых 4 моментов биномиального распределения с параметрами *n* и *p*.

```
double MathMomentsBinomial(
    const double n,           // параметр распределения (количество испытаний)
    const double p,           // параметр распределения (вероятность успеха для каждого испытания)
    double& mean,            // переменная для среднего значения
    double& variance,        // переменная для дисперсии
    double& skewness,         // переменная для коэффициента асимметрии
    double& kurtosis,         // переменная для коэффициента эксцесса
    int& error_code          // переменная для кода ошибки
);
```

### Параметры

*n*

[in] Параметр распределения (количество испытаний).

*p*

[in] Параметр распределения (вероятность успеха для каждого испытания).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает `true`, если расчет моментов произведен успешно, иначе `false`.

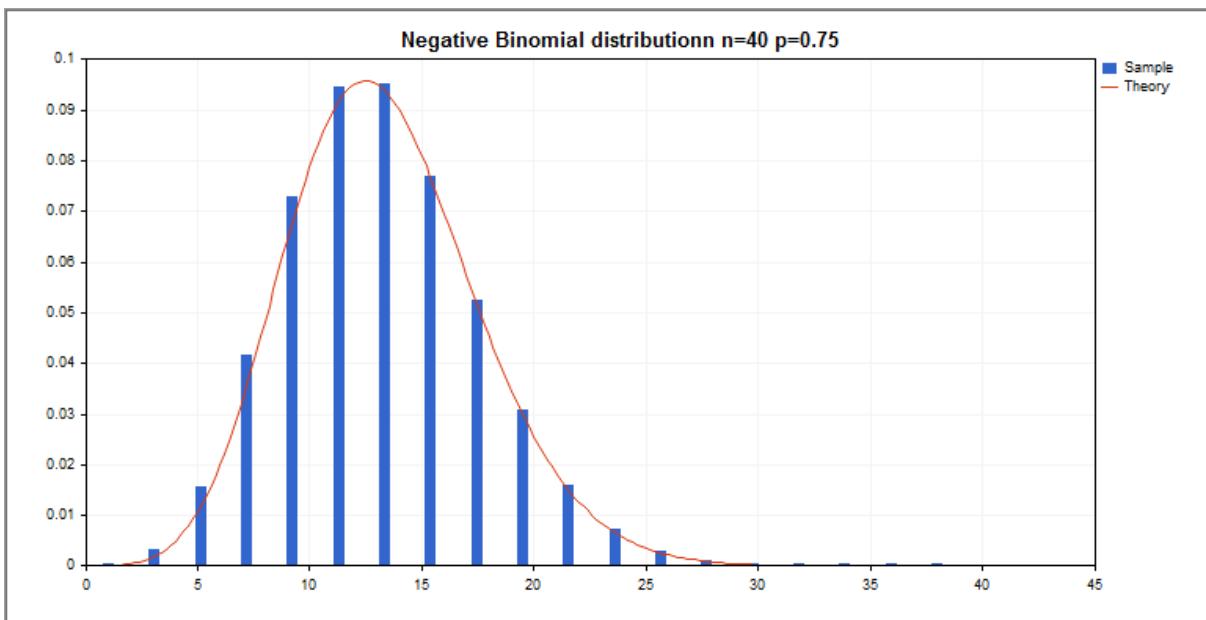
## Отрицательное биномиальное распределение

В данном разделе представлены функции для работы с отрицательным биномиальным распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по отрицательному биномиальному закону. Отрицательное биномиальное распределение описывается следующей формулой:

$$f_{\text{NegativeBinomial}}(x | r, p) = \frac{\Gamma(r+x)}{\Gamma(r)\Gamma(x+1)} p^r (1-p)^x$$

где:

- $x$  — значение случайной величины
- $r$  — количество успешных испытаний
- $p$  — вероятность успеха



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityNegativeBinomial</a>	Рассчитывает плотность вероятности биномиального распределения
<a href="#">MathCumulativeDistributionNegativeBinomial</a>	Рассчитывает значение функции биномиального распределения вероятностей
<a href="#">MathQuantileNegativeBinomial</a>	Рассчитывает значение обратной функции отрицательного биномиального распределения для заданной вероятности

<a href="#">MathRandomNegativeBinomial</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по отрицательному биномиальному закону
<a href="#">MathMomentsNegativeBinomial</a>	Рассчитывает теоретические численные значения первых 4 моментов отрицательного биномиального распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\NegativeBinomial.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double n_par=40;           // количество испытаний
input double p_par=0.75;          // вероятность успеха для каждого испытания
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;           // количество значений в выборке
    int ncells=19;            // количество интервалов в гистограмме
    double x[];               // центры интервалов гистограммы
    double y[];               // количество значений из выборки, попавших в интервал
    double data[];             // выборка случайных значений
    double max,min;           // максимальное и минимальное значения в выборке
//--- получим выборку из отрицательного биномиального распределения
    MathRandomNegativeBinomial(n_par,p_par,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим теоретически рассчитанные данные на интервале [min,max]
    double x2[];
    double y2[];
    MathSequence(0,n_par,1,x2);
    MathProbabilityDensityNegativeBinomial(x2,n_par,p_par,false,y2);
//--- масштабируем
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
```

```

for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- ВЫВОДИМ ГРАФИКИ
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Negative Binomial distributionn n=%G p=%G",n_p));
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set | 
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```

## MathProbabilityDensityNegativeBinomial

Рассчитывает значение функции вероятности (probability mass function) отрицательного биномиального распределения с параметрами *r* и *p* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNegativeBinomial(
    const double x,           // значение случайной величины (целочисленное)
    const double r,           // количество успешных испытаний
    const double p,           // вероятность успеха
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает значение функции вероятности (probability mass function) отрицательного биномиального распределения с параметрами *r* и *p* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityNegativeBinomial(
    const double x,           // значение случайной величины (целочисленное)
    const double r,           // количество успешных испытаний
    const double p,           // вероятность успеха
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает значение функции вероятности (probability mass function) отрицательного биномиального распределения с параметрами *r* и *p* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [dnbinom\(\)](#) в R.

```
bool MathProbabilityDensityNegativeBinomial(
    const double& x[],        // массив со значениями случайной величины
    const double r,            // количество успешных испытаний
    const double p,            // вероятность успеха
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает значение функции вероятности (probability mass function) отрицательного биномиального распределения с параметрами *r* и *p* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityNegativeBinomial(
    const double& x[],        // массив со значениями случайной величины
    const double r,            // количество успешных испытаний
    const double p,            // вероятность успеха
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x*[]

[in] Массив со значениями случайной величины.

*r*

[in] Количество успешных испытаний

*p*

[in] Вероятность успеха.

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode*=true, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result*[]

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionNegativeBinomial

Рассчитывает функцию распределения для отрицательного биномиального закона с параметрами *r* и *p* для случайной величины *x*. В случае ошибки возвращает `NaN`.

```
double MathCumulativeDistributionNegativeBinomial(
    const double x,           // значение случайной величины (целочисленное)
    const double r,           // количество успешных испытаний
    const double p,           // вероятность успеха
    const double tail,        // флаг расчета, если true, то рассчитывается вероятност
    const bool log_mode,      // флаг расчета логарифма значения, если log_mode=true
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает функцию распределения для отрицательного биномиального закона с параметрами *r* и *p* для случайной величины *x*. В случае ошибки возвращает `NaN`.

```
double MathCumulativeDistributionNegativeBinomial(
    const double x,           // значение случайной величины (целочисленное)
    const double r,           // количество успешных испытаний
    const double p,           // вероятность успеха
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает функцию распределения для отрицательного биномиального закона с параметрами *r* и *p* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [pweibull\(\)](#) в R.

```
bool MathCumulativeDistributionNegativeBinomial(
    const double& x[],        // массив со значениями случайной величины
    const double r,            // количество успешных испытаний
    const double p,            // вероятность успеха
    const double tail,         // флаг расчета, если true, то рассчитывается вероятност
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true
    double& result[]          // массив для значений функции вероятности
);
```

Рассчитывает функцию распределения для отрицательного биномиального закона с параметрами *r* и *p* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathCumulativeDistributionNegativeBinomial(
    const double& x[],        // массив со значениями случайной величины
    const double r,            // количество успешных испытаний
    const double p,            // вероятность успеха
    double& result[]          // массив для значений функции вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*x*

[in] Количество успешных испытаний.

*P*

[in] Вероятность успеха.

*tail*

[in] Флаг расчета, если true, то то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета, логарифма значения, если *log\_mode=true*, то рассчитывается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileNegativeBinomial

Рассчитывает обратное значение функции распределения для отрицательного биномиального закона с параметрами *г* и *р* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileNegativeBinomial(
    const double probability,           // значение вероятности появления случайной величины
    const double r,                    // количество успешных испытаний
    const double p,                    // вероятность успеха
    const bool tail,                 // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется для логарифма вероятности
    int& error_code                // переменная для записи кода ошибки
);
```

Рассчитывает обратное значение функции распределения для отрицательного биномиального закона с параметрами *г* и *р* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileNegativeBinomial(
    const double probability,           // значение вероятности появления случайной величины
    const double r,                    // количество успешных испытаний
    const double p,                    // вероятность успеха
    int& error_code                // переменная для записи кода ошибки
);
```

Рассчитывает обратное значение функции распределения для отрицательного биномиального закона с параметрами *г* и *р* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*. Аналог [qnbnom\(\)](#) в R.

```
double MathQuantileNegativeBinomial(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double r,                  // количество успешных испытаний
    const double p,                  // вероятность успеха
    const bool tail,                // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,             // флаг расчета, если log_mode=true, то расчет ведется для логарифма вероятности
    double& result[]                // массив со значениями квантилей
);
```

Рассчитывает обратное значение функции распределения для отрицательного биномиального закона с параметрами *г* и *р* для массива значений вероятности *probability[]*. В случае ошибки возвращает *false*.

```
bool MathQuantileNegativeBinomial(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double r,                  // количество успешных испытаний
    const double p,                  // вероятность успеха
    double& result[]                // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*r*

[in] Количество успешных испытаний.

*p*

[in] Вероятность успеха.

*tail*

[in] Флаг расчета, если *false*, то расчет ведется для вероятности 1.0-*probability*.

*log\_mode*

[in] Флаг расчета, если *log\_mode=true*, то расчет ведется для вероятности  $\text{Exp}(\text{probability})$ .

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomNegativeBinomial

Генерирует псевдослучайную величину, распределенную по закону отрицательного биномиального распределения с параметрами  $r$  и  $p$ . В случае ошибки возвращает `NaN`.

```
double MathRandomNegativeBinomial(
    const double r,           // количество успешных испытаний
    const double p,           // вероятность успеха
    int& error_code          // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону отрицательного биномиального распределения с параметрами  $r$  и  $p$ . В случае ошибки возвращает `false`. Аналог [rweibull\(\)](#) в R.

```
bool MathRandomNegativeBinomial(
    const double r,           // количество успешных испытаний
    const double p,           // вероятность успеха
    const int data_count,     // количество необходимых данных
    double& result[]          // массив со значениями псевдослучайных величин
);
```

### Параметры

$r$

[in] Количество успешных испытаний.

$p$

[in] Вероятность успеха.

`error_code`

[out] Переменная для записи кода ошибки.

`data_count`

[out] Количество необходимых данных.

`result[]`

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsNegativeBinomial

Рассчитывает теоретические численные значения первых 4 моментов отрицательного биномиального распределения с параметрами *r* и *p*.

```
double MathMomentsNegativeBinomial(
    const double r,                      // количество успешных испытаний
    const double p,                      // вероятность успеха
    double& mean,                     // переменная для среднего значения
    double& variance,                 // переменная для дисперсии
    double& skewness,                  // переменная для коэффициента асимметрии
    double& kurtosis,                  // переменная для коэффициента эксцесса
    int& error_code                  // переменная для кода ошибки
);
```

### Параметры

*r*

[in] Количество успешных испытаний.

*p*

[in] Вероятность успеха.

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает `true`, если расчет моментов произведен успешно, иначе `false`.

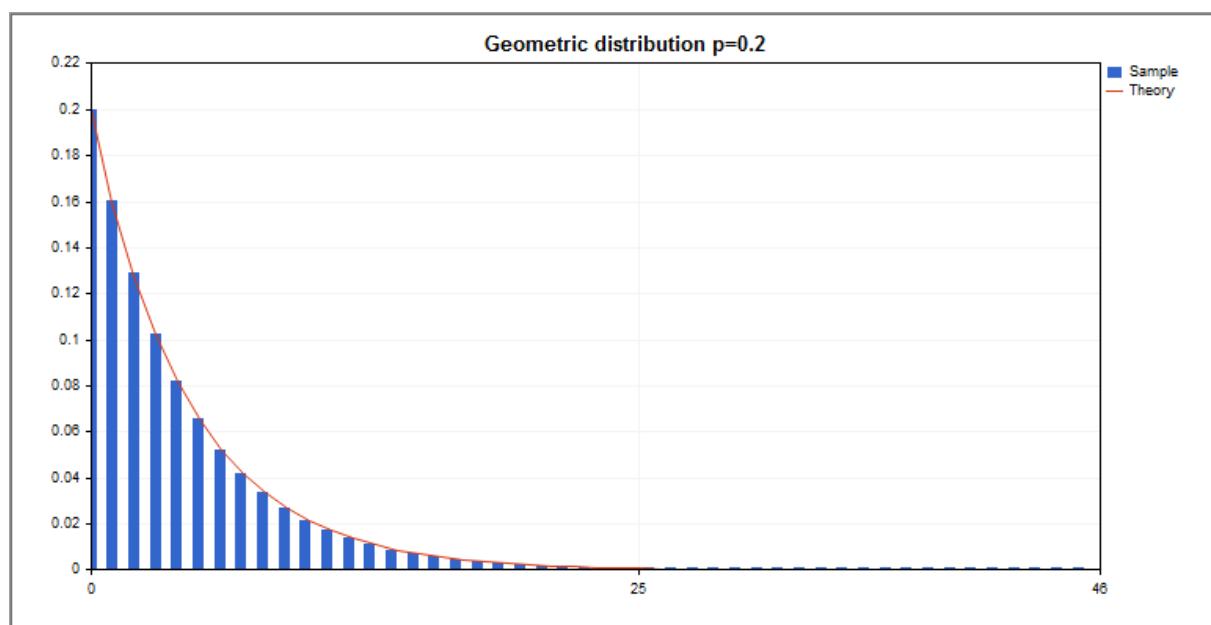
## Геометрическое распределение

В данном разделе представлены функции для работы с геометрическим распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по геометрическому закону. Геометрическое распределение описывается следующей формулой:

$$f_{\text{Geometric}}(x | p) = p(1-p)^x$$

где:

- $x$  — значение случайной величины (целочисленное)
- $p$  — вероятность появления события в одном опыте



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityGeometric</a>	Рассчитывает плотность вероятности геометрического распределения
<a href="#">MathCumulativeDistributionGeometric</a>	Рассчитывает значение функции вероятностей геометрического распределения
<a href="#">MathQuantileGeometric</a>	Рассчитывает значение обратной функции геометрического распределения для заданной вероятности
<a href="#">MathRandomGeometric</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по геометрическому закону

MathMomentsGeometric

Рассчитывает теоретические значения первых 4 численные моментов геометрического распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Geometric.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double p_par=0.2; // вероятность появления события в одном опыте
//+-----+
//| Script program start function | +-----+
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
long chart=0;
string name="GraphicNormal";
int n=1000000; // количество значений в выборке
int ncells=47; // количество интервалов в гистограмме
double x[]; // центры интервалов гистограммы
double y[]; // количество значений из выборки, попавших в интервал
double data[]; // выборка случайных значений
double max,min; // максимальное и минимальное значения в выборке
//--- получим выборку из геометрического распределения
MathRandomGeometric(p_par,n,data);
//--- рассчитаем данные для построения гистограммы
CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
double step;
GetMaxMinStepValues(max,min,step);
PrintFormat("max=%G min=%G",max,min);
//--- получим теоретически рассчитанные данные на интервале [min,max]
double x2[];
double y2[];
MathSequence(0,ncells,1,x2);
MathProbabilityDensityGeometric(x2,p_par,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
```

```

y[i]/=k;
//--- ВЫВОДИМ ГРАФИКИ
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Geometric distribution p=%G",p_par));
graphic.BackgroundMainSize(16);
//--- отключим автомасштабирование оси X
graphic.XAxis().AutoScale(false);
graphic.XAxis().Max(max);
graphic.XAxis().Min(min);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory");
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |+
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+i*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
}

```

```
        }
        return (true);
    }
//+-----+
//| Calculates values for sequence generation |
//+-----+
void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityGeometric

Рассчитывает значение функции вероятности (probability mass function) геометрического распределения с параметром *p* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityGeometric(
    const double x,           // значение случайной величины (целочисленное)
    const double p,           // параметр распределения (вероятность появления события)
    const bool log_mode,      // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает значение функции вероятности (probability mass function) геометрического распределения с параметром *p* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityGeometric(
    const double x,           // значение случайной величины (целочисленное)
    const double p,           // параметр распределения (вероятность появления события)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает значение функции вероятности (probability mass function) геометрического распределения с параметром *p* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [dgeom\(\)](#) в R.

```
bool MathProbabilityDensityGeometric(
    const double& x[],        // массив со значениями случайной величины
    const double p,            // параметр распределения (вероятность появления события)
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]          // массив для значений функции плотности вероятности
);
```

Рассчитывает значение функции вероятности (probability mass function) геометрического распределения с параметром *p* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityGeometric(
    const double& x[],        // массив со значениями случайной величины
    const double p,            // параметр распределения (вероятность появления события)
    double& result[]          // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*p*

[in] Параметр распределения (вероятность появления события в одном опыте).

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionGeometric

Рассчитывает функцию распределения для геометрического закона с параметром  $p$  для случайной величины  $x$ . В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionGeometric(
    const double x,           // значение случайной величины (целочисленное)
    const double p,           // параметр распределения (вероятность появления события)
    const double tail,        // флаг расчета, если true, то рассчитывается вероятность хвоста
    const bool log_mode,      // флаг расчета логарифма значения, если log_mode=true
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает функцию распределения для геометрического закона с параметром  $p$  для случайной величины  $x$ . В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionGeometric(
    const double x,           // значение случайной величины (целочисленное)
    const double p,           // параметр распределения (вероятность появления события)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает функцию распределения для геометрического закона с параметром  $p$  для массива случайных величин  $x[]$ . В случае ошибки возвращает `false`. Аналог [pgeom\(\)](#) в R.

```
bool MathCumulativeDistributionGeometric(
    const double& x[],        // массив со значениями случайной величины
    const double p,            // параметр распределения (вероятность появления события)
    const double tail,         // флаг расчета, если true, то рассчитывается вероятность хвоста
    const bool log_mode,       // флаг расчета логарифма значения, если log_mode=true
    double& result[]          // массив для значений функции вероятности
);
```

Рассчитывает функцию распределения для геометрического закона с параметром  $p$  для массива случайных величин  $x[]$ . В случае ошибки возвращает `false`.

```
bool MathCumulativeDistributionGeometric(
    const double& x[],        // массив со значениями случайной величины
    const double p,            // параметр распределения (вероятность появления события)
    double& result[]          // массив для значений функции вероятности
);
```

### Параметры

$x$

[in] Значение случайной величины.

$x[]$

[in] Массив со значениями случайной величины.

$p$

[in] Параметр распределения (вероятность появления события в одном опыте).

*tail*

[in] Флаг расчета, если *tail*=true, то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета логарифма значения, если *log\_mode*=true, то рассчитывается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции вероятности.

## MathQuantileGeometric

Рассчитывает обратное значение функции распределения для геометрического закона с параметром *p* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileGeometric(
    const double probability,           // значение вероятности появления случайной величины
    const double p,                     // параметр распределения (вероятность появления события)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает обратное значение функции распределения для геометрического закона с параметром *p* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileGeometric(
    const double probability,           // значение вероятности появления случайной величины
    const double p,                     // параметр распределения (вероятность появления события)
    int& error_code                  // переменная для записи кода ошибки
);
```

Рассчитывает обратное значение функции распределения для геометрического закона с параметром *p* для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`. Аналог [qgeom\(\)](#) в R.

```
double MathQuantileGeometric(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double p,                     // параметр распределения (вероятность появления события)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает обратное значение функции распределения для геометрического закона с параметром *p* для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`.

```
bool MathQuantileGeometric(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double p,                     // параметр распределения (вероятность появления события)
    double& result[]                  // массив со значениями квантилей
);
```

### Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*p*

[in] Параметр распределения (вероятность появления события в одном опыте).

*tail*

[in] Флаг расчета, если `false`, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если `log_mode=true`, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomGeometric

Генерирует псевдослучайную величину, распределенную по закону геометрического распределения с параметром p. В случае ошибки возвращает [NaN](#).

```
double MathRandomGeometric(
    const double p, // параметр распределения (вероятность появления события)
    int& error_code // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону геометрического распределения с параметром p. В случае ошибки возвращает false. Аналог [rgeom\(\)](#) в R.

```
bool MathRandomGeometric(
    const double p, // параметр распределения (вероятность появления события)
    const int data_count, // количество необходимых данных
    double& result[] // массив со значениями псевдослучайных величин
);
```

### Параметры

*p*

[in] Параметр распределения (вероятность появления события в одном опыте).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsGeometric

Рассчитывает теоретические численные значения первых 4 моментов геометрического распределения с параметром *p*.

```
double MathMomentsGeometric(
    const double p, // параметр распределения (вероятность появления события)
    double& mean, // переменная для среднего значения
    double& variance, // переменная для дисперсии
    double& skewness, // переменная для коэффициента асимметрии
    double& kurtosis, // переменная для коэффициента эксцесса
    int& error_code // переменная для кода ошибки
);
```

### Параметры

*p*

[in] Параметр распределения (вероятность появления события в одном опыте).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает `true`, если расчет моментов произведен успешно, иначе `false`.

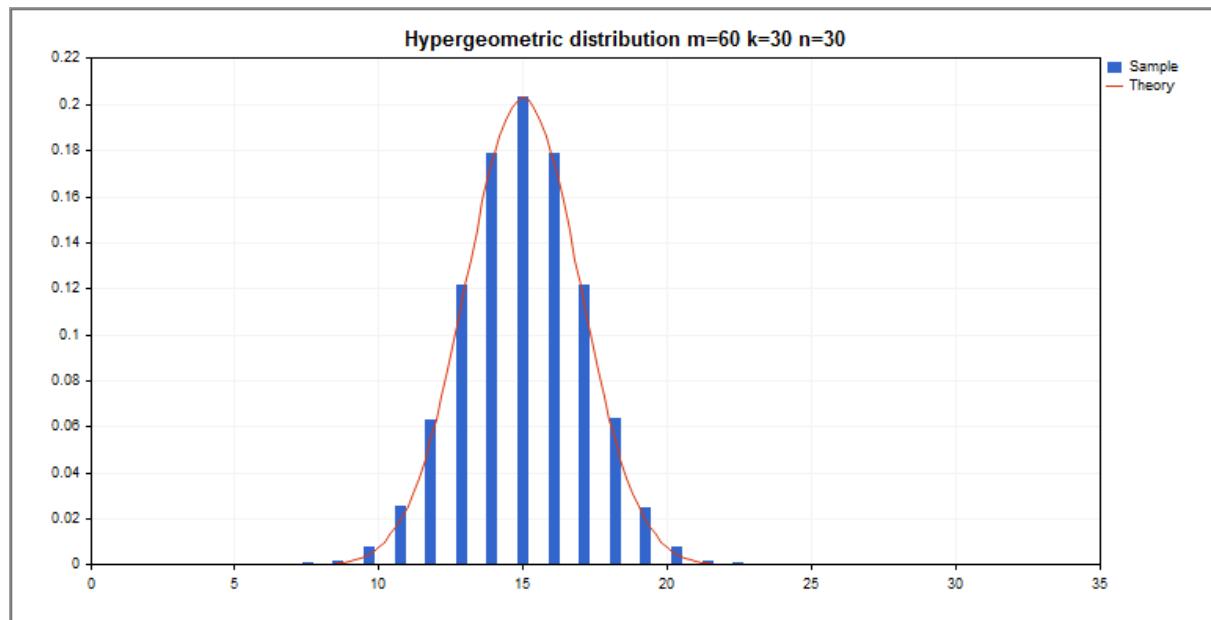
## Гипергеометрическое распределение

В данном разделе представлены функции для работы с гипергеометрическим распределением. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по гипергеометрическому закону. Гипергеометрическое распределение описывается следующей формулой:

$$f_{\text{Hypergeometric}}(x | m, k, n) = \frac{\binom{k}{x} \binom{m-k}{n-x}}{\binom{m}{n}}$$

где:

- $x$  — значение случайной величины (целочисленное)
- $m$  — общее количество объектов
- $k$  — количество объектов с желаемой характеристикой
- $n$  — количество взятых объектов



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityHypergeometric</a>	Рассчитывает плотность вероятности гипергеометрического распределения
<a href="#">MathCumulativeDistributionHypergeometric</a>	Рассчитывает значение функции вероятностей гипергеометрического распределения

<a href="#"><u>MathQuantileHypergeometric</u></a>	Рассчитывает значение обратной функции гипергеометрического распределения для заданной вероятности
<a href="#"><u>MathRandomHypergeometric</u></a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по гипергеометрическому закону
<a href="#"><u>MathMomentsHypergeometric</u></a>	Рассчитывает теоретические численные значения первых 4 моментов гипергеометрического распределения

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Hypergeometric.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double m_par=60;           // общее количество объектов
input double k_par=30;           // количество объектов с желаемой характеристикой
input double n_par=30;           // количество взятых объектов
//+-----+
//| Script program start function | 
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathStrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=1000000;                 // количество значений в выборке
    int ncells=15;                  // количество интервалов в гистограмме
    double x[];                     // центры интервалов гистограммы
    double y[];                     // количество значений из выборки, попавших в интервал
    double data[];                  // выборка случайных значений
    double max,min;                // максимальное и минимальное значения в выборке
//--- получим выборку из гипергеометрического распределения
    MathRandomHypergeometric(m_par,k_par,n_par,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
    double step;
    GetMaxMinStepValues(max,min,step);
    PrintFormat("max=%G min=%G",max,min);
//--- получим теоретически рассчитанные данные на интервале [min,max]
```

```

double x2[];
double y2[];

MathSequence(0,n_par,1,x2);
MathProbabilityDensityHypergeometric(x2,m_par,k_par,n_par,false,y2);
//--- масштабируем
double theor_max=y2[ArrayMaximum(y2)];
double sample_max=y[ArrayMaximum(y)];
double k=sample_max/theor_max;
for(int i=0; i<ncells; i++)
    y[i]/=k;
//--- выводим графики
CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Hypergeometric distribution m=%G k=%G n=%G",m_
graphic.BackgroundMainSize(16);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//+-----+
//| Calculate frequencies for data set |
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
}

```

```
for(int i=0; i<size; i++)
{
    int ind=int((data[i]-minv)/width);
    if(ind>=cells) ind=cells-1;
    frequency[ind]++;
}
return (true);
}

//+-----+
//| Calculates values for sequence generation |
//+-----+

void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityHypergeometric

Рассчитывает значение функции вероятности (probability mass function) гипергеометрического распределения с параметрами  $m$ ,  $k$  и  $n$  для случайной величины  $x$ . В случае ошибки возвращает ***NaN***.

```
double MathProbabilityDensityHypergeometric(
    const double x,                      // значение случайной величины (целочисленное)
    const double m,                      // общее количество объектов (целочисленное)
    const double k,                      // количество объектов с желаемой характеристикой (целочисленное)
    const double n,                      // количество взятых объектов (целочисленное)
    const bool log_mode,                // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code                   // переменная для записи кода ошибки
);
```

Рассчитывает значение функции вероятности (probability mass function) гипергеометрического распределения с параметрами  $m$ ,  $k$  и  $n$  для случайной величины  $x$ . В случае ошибки возвращает ***NaN***.

```
double MathProbabilityDensityHypergeometric(
    const double x,                      // значение случайной величины (целочисленное)
    const double m,                      // общее количество объектов (целочисленное)
    const double k,                      // количество объектов с желаемой характеристикой (целочисленное)
    const double n,                      // количество взятых объектов (целочисленное)
    int& error_code                   // переменная для записи кода ошибки
);
```

Рассчитывает значение функции вероятности (probability mass function) гипергеометрического распределения с параметрами  $m$ ,  $k$  и  $n$  для массива случайных величин  $x[]$ . В случае ошибки возвращает ***false***. Аналог [dhyper\(\)](#) в R.

```
bool MathProbabilityDensityHypergeometric(
    const double& x[],                 // массив со значениями случайной величины
    const double m,                      // общее количество объектов (целочисленное)
    const double k,                      // количество объектов с желаемой характеристикой (целочисленное)
    const double n,                      // количество взятых объектов (целочисленное)
    const bool log_mode,                // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]                  // массив для значений функции плотности вероятности
);
```

Рассчитывает значение функции вероятности (probability mass function) гипергеометрического распределения с параметрами  $m$ ,  $k$  и  $n$  для массива случайных величин  $x[]$ . В случае ошибки возвращает ***false***.

```
bool MathProbabilityDensityHypergeometric(
    const double& x[],                 // массив со значениями случайной величины
    const double m,                      // общее количество объектов (целочисленное)
    const double k,                      // количество объектов с желаемой характеристикой (целочисленное)
    const double n,                      // количество взятых объектов (целочисленное)
    double& result[]                  // массив для значений функции плотности вероятности
);
```

```
) ;
```

## Параметры

*x*

[in] Значение случайной величины.

*x* [ ]

[in] Массив со значениями случайной величины.

*m*

[in] Общее количество объектов (целочисленное).

*k*

[in] Количество объектов с желаемой характеристикой (целочисленное).

*n*

[in] Количество взятых объектов (целочисленное).

*log\_mode*

[in] Флаг расчета логарифма значения. Если *log\_mode=true*, то возвращается натуральный логарифм плотности вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result* [ ]

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionHypergeometric

Рассчитывает значение функции распределения для гипергеометрического закона с параметрами m, k и n для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionHypergeometric(
    const double x, // значение случайной величины (целочисленное)
    const double m, // общее количество объектов (целочисленное)
    const double k, // количество объектов с желаемой характеристикой (целочисленное)
    const double n, // количество взятых объектов (целочисленное)
    const double tail, // флаг расчета, если true, то рассчитывается вероятность х <= x
    const bool log_mode, // флаг расчета логарифма значения, если log_mode=true
    int& error_code // переменная для записи кода ошибки
);
```

Рассчитывает значение функции распределения для гипергеометрического закона с параметрами m, k и n для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionHypergeometric(
    const double x, // значение случайной величины (целочисленное)
    const double m, // общее количество объектов (целочисленное)
    const double k, // количество объектов с желаемой характеристикой (целочисленное)
    const double n, // количество взятых объектов (целочисленное)
    int& error_code // переменная для записи кода ошибки
);
```

Рассчитывает значение функции распределения для гипергеометрического закона с параметрами m, k и n для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [dhyper\(\)](#) в R.

```
bool MathCumulativeDistributionHypergeometric(
    const double& x[], // массив со значениями случайной величины
    const double m, // общее количество объектов (целочисленное)
    const double k, // количество объектов с желаемой характеристикой (целочисленное)
    const double n, // количество взятых объектов (целочисленное)
    const double tail, // флаг расчета, если true, то рассчитывается вероятность х <= x
    const bool log_mode, // флаг расчета логарифма значения, если log_mode=true
    double& result[] // массив для значений функции распределения
);
```

Рассчитывает значение функции распределения для гипергеометрического закона с параметрами m, k и n для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathCumulativeDistributionHypergeometric(
    const double& x[], // массив со значениями случайной величины
    const double m, // общее количество объектов (целочисленное)
    const double k, // количество объектов с желаемой характеристикой (целочисленное)
    const double n, // количество взятых объектов (целочисленное)
    double& result[] // массив для значений функции распределения
);
```

## Параметры

*x*

[in] Значение случайной величины.

*x [ ]*

[in] Массив со значениями случайной величины.

*m*

[in] Общее количество объектов (целочисленное).

*k*

[in] Количество объектов с желаемой характеристикой (целочисленное).

*n*

[in] Количество взятых объектов (целочисленное).

*tail*

[in] Флаг расчета, если true, то то рассчитывается вероятность того, что случайная величина не превысит *x*.

*log\_mode*

[in] Флаг расчета, логарифма значения, если log\_mode=true, то рассчитывается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result [ ]*

[out] Массив для значений функции распределения.

## MathQuantileHypergeometric

Рассчитывает обратное значение функции распределения для гипергеометрического закона с параметрами *m*, *k* и *n* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileHypergeometric(
    const double probability,           // значение вероятности появления случайной величины
    const double m,                    // общее количество объектов (целочисленное)
    const double k,                    // количество объектов с желаемой характеристикой (целочисленное)
    const double n,                    // количество взятых объектов (целочисленное)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает обратное значение функции распределения для гипергеометрического закона с параметрами *m*, *k* и *n* для вероятности *probability*. В случае ошибки возвращает [NaN](#).

```
double MathQuantileHypergeometric(
    const double probability,           // значение вероятности появления случайной величины
    const double m,                    // общее количество объектов (целочисленное)
    const double k,                    // количество объектов с желаемой характеристикой (целочисленное)
    const double n,                    // количество взятых объектов (целочисленное)
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает обратное значение функции распределения для гипергеометрического закона с параметрами *m*, *k* и *n* для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`. Аналог [qhyper\(\)](#) в R.

```
double MathQuantileHypergeometric(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double m,                  // общее количество объектов (целочисленное)
    const double k,                  // количество объектов с желаемой характеристикой (целочисленное)
    const double n,                  // количество взятых объектов (целочисленное)
    const bool tail,                // флаг расчета, если false, то расчет ведется для верхней квантилью
    const bool log_mode,             // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    double& result[]                // массив со значениями квантилей
);
```

Рассчитывает обратное значение функции распределения для гипергеометрического закона с параметрами *m*, *k* и *n* для массива значений вероятности *probability[]*. В случае ошибки возвращает `false`.

```
bool MathQuantileHypergeometric(
    const double& probability[],     // массив со значениями вероятностей случайной величины
    const double m,                  // общее количество объектов (целочисленное)
    const double k,                  // количество объектов с желаемой характеристикой (целочисленное)
    const double n,                  // количество взятых объектов (целочисленное)
    double& result[]                // массив со значениями квантилей
);
```

```
) ;
```

## Параметры

*probability*

[in] Значение вероятности случайной величины.

*probability[]*

[in] Массив со значениями вероятностей случайной величины.

*m*

[in] Общее количество объектов (целочисленное).

*k*

[in] Количество объектов с желаемой характеристикой (целочисленное).

*n*

[in] Количество взятых объектов (целочисленное).

*tail*

[in] Флаг расчета, если *tail=false*, то расчет ведется для вероятности 1.0-*probability*.

*log\_mode*

[in] Флаг расчета, если *log\_mode=true*, то расчет ведется для вероятности Exp(*probability*).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomHypergeometric

Генерирует псевдослучайную величину, распределенную по закону гипергеометрического распределения с параметрами *m*, *n* и *k*. В случае ошибки возвращает [NaN](#).

```
double MathRandomHypergeometric(
    const double m, // общее количество объектов (целочисленное)
    const double k, // количество объектов с желаемой характеристикой (целочисленное)
    const double n, // количество взятых объектов (целочисленное)
    int& error_code // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону гипергеометрического распределения с параметрами *m*, *n* и *k*. В случае ошибки возвращает `false`. Аналог [rgeom\(\)](#) в R.

```
bool MathRandomHypergeometric(
    const double m, // общее количество объектов (целочисленное)
    const double k, // количество объектов с желаемой характеристикой (целочисленное)
    const double n, // количество взятых объектов (целочисленное)
    const int data_count, // количество необходимых данных
    double& result[] // массив со значениями псевдослучайных величин
);
```

### Параметры

*m*

[in] Общее количество объектов (целочисленное).

*k*

[in] Количество объектов с желаемой характеристикой (целочисленное).

*n*

[in] Количество взятых объектов (целочисленное).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsHypergeometric

Рассчитывает теоретические численные значения первых 4 моментов гипергеометрического распределения с параметрами *m*, *n* и *k*.

```
double MathMomentsHypergeometric(
    const double m, // общее количество объектов (целочисленное)
    const double k, // количество объектов с желаемой характеристикой (целочисленное)
    const double n, // количество взятых объектов (целочисленное)
    double& mean, // переменная для среднего значения
    double& variance, // переменная для дисперсии
    double& skewness, // переменная для коэффициента асимметрии
    double& kurtosis, // переменная для коэффициента эксцесса
    int& error_code // переменная для кода ошибки
);
```

### Параметры

*m*

[in] Общее количество объектов (целочисленное).

*k*

[in] Количество объектов с желаемой характеристикой (целочисленное).

*n*

[in] Количество взятых объектов (целочисленное).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает `true`, если расчет моментов произведен успешно, иначе `false`.

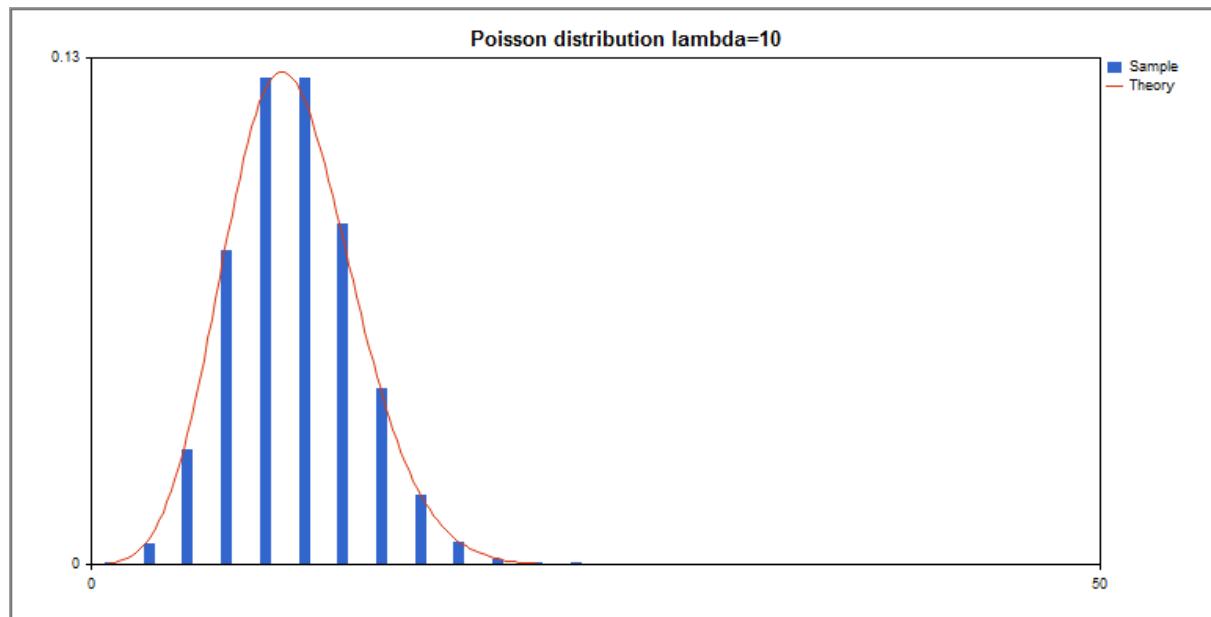
## Распределение Пуассона

В данном разделе представлены функции для работы с распределением Пуассона. Они позволяют производить расчет плотности, вероятности, квантилей и генерировать псевдослучайные числа, распределенные по закону Пуассона. Распределение Пуассона описывается следующей формулой:

$$f_{Poisson}(x | \lambda) = \frac{\lambda^x}{x!} e^{-\lambda}$$

где:

- $x$  — значение случайной величины
- $\lambda$  — параметр распределения (mean)



Помимо расчета отдельных случайных величин, реализована возможность работы с их массивами.

Функция	Описание
<a href="#">MathProbabilityDensityPoisson</a>	Рассчитывает плотность вероятности распределения Пуассона
<a href="#">MathCumulativeDistributionPoisson</a>	Рассчитывает значение функции распределения вероятностей Пуассона
<a href="#">MathQuantilePoisson</a>	Рассчитывает значение обратной функции распределения Пуассона для заданной вероятности
<a href="#">MathRandomPoisson</a>	Генерирует псевдослучайную величину/массив псевдослучайных величин, распределенных по закону Пуассона
<a href="#">MathMomentsPoisson</a>	Рассчитывает теоретические численные значения первых 4 моментов распределения

## Пуассона

**Пример:**

```
#include <Graphics\Graphic.mqh>
#include <Math\Stat\Poisson.mqh>
#include <Math\Stat\Math.mqh>
#property script_show_inputs
//--- input parameters
input double lambda_par=10;           // параметр распределения (mean)
//+-----+
//| Script program start function          |
//+-----+
void OnStart()
{
//--- отключим показ ценового графика
    ChartSetInteger(0,CHART_SHOW,false);
//--- инициализируем генератор случайных чисел
    MathSrand(GetTickCount());
//--- сгенерируем выборку случайной величины
    long chart=0;
    string name="GraphicNormal";
    int n=100000;           // количество значений в выборке
    int ncells=13;          // количество интервалов в гистограмме
    double x[];             // центры интервалов гистограммы
    double y[];             // количество значений из выборки, попавших в интервал
    double data[];          // выборка случайных значений
    double max,min;         // максимальное и минимальное значения в выборке
//--- получим выборку из распределения Пуассона
    MathRandomPoisson(lambda_par,n,data);
//--- рассчитаем данные для построения гистограммы
    CalculateHistogramArray(data,x,y,max,min,ncells);
//--- получим границы последовательности и шаг для построения теоретической кривой
    double step;
    GetMaxMinStepValues(max,min,step);
    PrintFormat("max=%G min=%G",max,min);
//--- получим теоретически рассчитанные данные на интервале [min,max]
    double x2[];
    double y2[];
    MathSequence(0,int(MathCeil(max)),1,x2);
    MathProbabilityDensityPoisson(x2,lambda_par,false,y2);
//--- масштабируем
    double theor_max=y2[ArrayMaximum(y2)];
    double sample_max=y[ArrayMaximum(y)];
    double k=sample_max/theor_max;
    for(int i=0; i<ncells; i++)
        y[i]/=k;
//--- выводим графики
```

```

CGraphic graphic;
if(ObjectFind(chart,name)<0)
    graphic.Create(chart,name,0,0,0,780,380);
else
    graphic.Attach(chart,name);
graphic.BackgroundMain(StringFormat("Poisson distribution lambda=%G",lambda_par));
graphic.BackgroundMainSize(16);
//--- отключим автомасштабирование оси Y
graphic.YAxis().AutoScale(false);
graphic.YAxis().Max(NormalizeDouble(theor_max,2));
graphic.YAxis().Min(0);
//--- plot all curves
graphic.CurveAdd(x,y,CURVE_HISTOGRAM,"Sample").HistogramWidth(6);
//--- а теперь построим теоретическую кривую плотности распределения
graphic.CurveAdd(x2,y2,CURVE_LINES,"Theory").LinesSmooth(true);
graphic.CurvePlotAll();
//--- plot all curves
graphic.Update();
}
//----------------------------------------------------------------------------------------------------------------+
//| Calculate frequencies for data set |+
//+-----+
bool CalculateHistogramArray(const double &data[],double &intervals[],double &frequency,
                             double &maxv,double &minv,const int cells=10)
{
    if(cells<=1) return (false);
    int size=ArraySize(data);
    if(size<cells*10) return (false);
    minv=data[ArrayMinimum(data)];
    maxv=data[ArrayMaximum(data)];
    double range=maxv-minv;
    double width=range/cells;
    if(width==0) return false;
    ArrayResize(intervals,cells);
    ArrayResize(frequency,cells);
//--- зададим центры интервалов
    for(int i=0; i<cells; i++)
    {
        intervals[i]=minv+(i+0.5)*width;
        frequency[i]=0;
    }
//--- заполним частоты попадания в интервал
    for(int i=0; i<size; i++)
    {
        int ind=int((data[i]-minv)/width);
        if(ind>=cells) ind=cells-1;
        frequency[ind]++;
    }
    return (true);
}

```

```
}

//+-----+
//| Calculates values for sequence generation |
//+-----+

void GetMaxMinStepValues(double &maxv,double &minv,double &stepv)
{
//--- вычислим абсолютный размах последовательности, чтобы получить точность нормализации
    double range=MathAbs(maxv-minv);
    int degree=(int)MathRound(MathLog10(range));
//--- нормализуем макс. и мин. значения с заданной точностью
    maxv=NormalizeDouble(maxv,degree);
    minv=NormalizeDouble(minv,degree);
//--- шаг генерации последовательности также зададим от заданной точности
    stepv=NormalizeDouble(MathPow(10,-degree),degree);
    if((maxv-minv)/stepv<10)
        stepv/=10.;
}
```

## MathProbabilityDensityPoisson

Рассчитывает значение функции вероятности (probability mass function) распределения Пуассона с параметром *lambda* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityPoisson(
    const double x,           // значение случайной величины (целочисленное)
    const double lambda,       // параметр распределения (mean)
    const bool log_mode,       // расчет логарифма значения, если log_mode=true, то возвращается логарифм
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает значение функции вероятности (probability mass function) распределения Пуассона с параметром *lambda* для случайной величины *x*. В случае ошибки возвращает [NaN](#).

```
double MathProbabilityDensityPoisson(
    const double x,           // значение случайной величины (целочисленное)
    const double lambda,        // параметр распределения (mean)
    int& error_code           // переменная для записи кода ошибки
);
```

Рассчитывает значение функции вероятности (probability mass function) распределения Пуассона с параметром *lambda* для массива случайных величин *x[]*. В случае ошибки возвращает `false`. Аналог [dhyper\(\)](#) в R.

```
bool MathProbabilityDensityPoisson(
    const double& x[],          // массив со значениями случайной величины
    const double lambda,         // параметр распределения (mean)
    const bool log_mode,         // флаг расчета логарифма значения, если log_mode=true, то возвращается логарифм
    double& result[]            // массив для значений функции плотности вероятности
);
```

Рассчитывает значение функции вероятности (probability mass function) распределения Пуассона с параметром *lambda* для массива случайных величин *x[]*. В случае ошибки возвращает `false`.

```
bool MathProbabilityDensityPoisson(
    const double& x[],          // массив со значениями случайной величины
    const double lambda,         // параметр распределения (mean)
    double& result[]             // массив для значений функции плотности вероятности
);
```

### Параметры

*x*

[in] Значение случайной величины.

*x[]*

[in] Массив со значениями случайной величины.

*lambda*

[in] Параметр распределения (mean).

*log\_mode*

[in] Флаг расчета логарифма значения. Если `log_mode=true`, то возвращается натуральный логарифм плотности вероятности.

`error_code`

[out] Переменная для записи кода ошибки.

`result[]`

[out] Массив для значений функции плотности вероятности.

## MathCumulativeDistributionPoisson

Рассчитывает значение функции распределения Пуассона с параметром lambda для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionPoisson(
    const double x,           // значение случайной величины (целочисленное)
    const double lambda,      // параметр распределения (mean)
    const double tail,        // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,      // флаг расчета логарифма значения, если log_mode=true
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает значение функции распределения Пуассона с параметром lambda для случайной величины x. В случае ошибки возвращает [NaN](#).

```
double MathCumulativeDistributionPoisson(
    const double x,           // значение случайной величины (целочисленное)
    const double lambda,       // параметр распределения (mean)
    int& error_code          // переменная для записи кода ошибки
);
```

Рассчитывает значение функции распределения Пуассона с параметром lambda для массива случайных величин x[]. В случае ошибки возвращает false. Аналог [dhyper\(\)](#) в R.

```
bool MathCumulativeDistributionPoisson(
    const double& x[],         // массив со значениями случайной величины
    const double lambda,        // параметр распределения (mean)
    const double tail,          // флаг расчета, если true, то рассчитывается вероятность
    const bool log_mode,        // флаг расчета логарифма значения, если log_mode=true
    double& result[]           // массив для значений функции распределения
);
```

Рассчитывает значение функции распределения Пуассона с параметром lambda для массива случайных величин x[]. В случае ошибки возвращает false.

```
bool MathCumulativeDistributionPoisson(
    const double& x[],         // массив со значениями случайной величины
    const double lambda,        // параметр распределения (mean)
    double& result[]           // массив для значений функции распределения
);
```

### Параметры

x

[in] Значение случайной величины.

x[]

[in] Массив со значениями случайной величины.

lambda

[in] Параметр распределения (mean).

*tail*

[in] Флаг расчета, если true, то то рассчитывается вероятность того, что случайная величина не превысит x.

*log\_mode*

[in] Флаг расчета, логарифма значения, если log\_mode=true, то рассчитывается натуральный логарифм вероятности.

*error\_code*

[out] Переменная для записи кода ошибки.

*result[]*

[out] Массив для значений функции распределения.

## MathQuantilePoisson

Рассчитывает обратное значение функции распределения Пуассона с параметром `lambda` для вероятности `probability`. В случае ошибки возвращает `NaN`.

```
double MathQuantilePoisson(
    const double probability,           // значение вероятности появления случайной величины
    const double lambda,                // параметр распределения (mean)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    int& error_code                 // переменная для записи кода ошибки
);
```

Рассчитывает обратное значение функции распределения Пуассона с параметром `lambda` для вероятности `probability`. В случае ошибки возвращает `NaN`.

```
double MathQuantilePoisson(
    const double probability,           // значение вероятности появления случайной величины
    const double lambda,                // параметр распределения (mean)
    int& error_code                  // переменная для записи кода ошибки
);
```

Рассчитывает обратное значение функции распределения Пуассона с параметром `lambda` для массива значений вероятности `probability[]`. В случае ошибки возвращает `false`. Аналог [qhyper\(\)](#) в R.

```
double MathQuantilePoisson(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double lambda,                // параметр распределения (mean)
    const bool tail,                  // флаг расчета, если false, то расчет ведется для вероятности 1 - probability
    const bool log_mode,              // флаг расчета, если log_mode=true, то расчет ведется в логарифмическом масштабе
    double& result[]                 // массив со значениями квантилей
);
```

Рассчитывает обратное значение функции распределения Пуассона с параметром `lambda` для массива значений вероятности `probability[]`. В случае ошибки возвращает `false`.

```
bool MathQuantilePoisson(
    const double& probability[],      // массив со значениями вероятностей случайной величины
    const double lambda,                // параметр распределения (mean)
    double& result[]                  // массив со значениями квантилей
);
```

### Параметры

`probability`

[in] Значение вероятности случайной величины.

`probability[]`

[in] Массив со значениями вероятностей случайной величины.

`lambda`

[in] Параметр распределения (mean).

*tail*

[in] Флаг расчета, если *tail=false*, то расчет ведется для вероятности 1.0-probability.

*log\_mode*

[in] Флаг расчета, если *log\_mode=true*, то расчет ведется для вероятности Exp(probability).

*error\_code*

[out] Переменная для получения кода ошибки.

*result[]*

[out] Массив со значениями квантилей.

## MathRandomPoisson

Генерирует псевдослучайную величину, распределенную по закону распределения Пуассона с параметром lambda. В случае ошибки возвращает [NaN](#).

```
double MathRandomPoisson(
    const double lambda,           // параметр распределения (mean)
    int& error_code               // переменная для записи кода ошибки
);
```

Генерирует псевдослучайные величины, распределенные по закону распределения Пуассона с параметром lambda. В случае ошибки возвращает false. Аналог [rgeom\(\)](#) в R.

```
bool MathRandomPoisson(
    const double lambda,           // параметр распределения (mean)
    const int data_count,          // количество необходимых данных
    double& result[]              // массив со значениями псевдослучайных величин
);
```

### Параметры

*lambda*

[in] Параметр распределения (mean).

*error\_code*

[out] Переменная для записи кода ошибки.

*data\_count*

[out] Количество необходимых данных.

*result[]*

[out] Массив для получения значений псевдослучайных величин.

## MathMomentsPoisson

Рассчитывает теоретические численные значения первых 4 моментов распределения Пуассона с параметром lambda.

```
double MathMomentsPoisson(
    const double lambda,           // параметр распределения (mean)
    double& mean,                 // переменная для среднего значения
    double& variance,            // переменная для дисперсии
    double& skewness,             // переменная для коэффициента асимметрии
    double& kurtosis,             // переменная для коэффициента эксцесса
    int& error_code              // переменная для кода ошибки
);
```

### Параметры

*lambda*

[in] Параметр распределения (mean).

*mean*

[out] Переменная для получения среднего значения.

*variance*

[out] Переменная для получения дисперсии.

*skewness*

[out] Переменная для получения коэффициента асимметрии.

*kurtosis*

[out] Переменная для получения коэффициента эксцесса.

*error\_code*

[out] Переменная для получения кода ошибки.

### Возвращаемое значение

Возвращает true, если расчет моментов произведен успешно, иначе false.

## Вспомогательные функции

Группа функций, выполняющих основные математические операции: расчет гамма-функции, бета-функции, факториала, экспоненты, логарифмов с различными основаниями, квадратного корня и т.п.

Предоставлены возможности обработки как отдельных числовых значений (вещественных и целочисленных), так и их массивов (с записью результатов в отдельный или в исходный массив).

Функция	Описание
<a href="#">MathRandomNonZero</a>	Возвращает случайное число в диапазоне от 0.0 до 1.0 с плавающей запятой.
<a href="#">MathMoments</a>	Рассчитывает первые 4 момента элементов массива: среднее, дисперсию, коэффициент асимметрии, коэффициент эксцесса.
<a href="#">MathPowInt</a>	Возводит число в указанную целочисленную степень.
<a href="#">MathFactorial</a>	Рассчитывает факториал заданного целого числа.
<a href="#">MathTrunc</a>	Рассчитывает целую часть заданного числа или элементов массива.
<a href="#">MathRound</a>	Округляет число или массив чисел до заданного количества дробных разрядов.
<a href="#">MathArctan2</a>	Рассчитывает угол, тангенс которого равен отношению двух указанных чисел в диапазоне [- $\pi$ , $\pi$ ].
<a href="#">MathGamma</a>	Вычисляет значение гамма-функции.
<a href="#">MathGammaLog</a>	Вычисляет логарифм гамма-функции.
<a href="#">MathBeta</a>	Вычисляет значение бета-функции.
<a href="#">MathBetaLog</a>	Вычисляет значение логарифма бета-функции.
<a href="#">MathBetaIncomplete</a>	Рассчитывает значение неполной бета-функции.
<a href="#">MathGammalncomplete</a>	Рассчитывает значение неполной гамма-функции.
<a href="#">MathBinomialCoefficient</a>	Рассчитывает биномиальный коэффициент.
<a href="#">MathBinomialCoefficientLog</a>	Рассчитывает логарифм биномиального коэффициента.
<a href="#">MathHypergeometric2F2</a>	Рассчитывает значение гипергеометрической функции.

<a href="#">MathSequence</a>	Формирует последовательность на основе значений: первый элемент, последний элемент, шаг последовательности.
<a href="#">MathSequenceByCount</a>	Формирует последовательность на основе значений: первый элемент, последний элемент, количество элементов последовательности.
<a href="#">MathReplicate</a>	Формирует повторяющуюся последовательность значений.
<a href="#">MathReverse</a>	Формирует массив значений с обратным порядком элементов.
<a href="#">MathIdentical</a>	Сравнивает два массива значений и возвращает true, если совпадают все элементы.
<a href="#">MathUnique</a>	Формирует массив только с неповторяющимися значениями.
<a href="#">MathQuickSortAscending</a>	Функция для восходящей сортировки.
<a href="#">MathQuickSortDescending</a>	Функция для нисходящей сортировки.
<a href="#">MathQuickSort</a>	Функция для сортировки.
<a href="#">MathOrder</a>	Формирует массив с перестановкой в соответствии с порядком элементов массива после сортировки.
<a href="#">MathBitwiseNot</a>	Рассчитывает результат бинарной операции NOT для элементов массива.
<a href="#">MathBitwiseAnd</a>	Рассчитывает результат бинарной операции AND для элементов массивов.
<a href="#">MathBitwiseOr</a>	Рассчитывает результат бинарной операции OR для элементов массивов.
<a href="#">MathBitwiseXor</a>	Рассчитывает результат бинарной операции XOR для элементов массивов.
<a href="#">MathBitwiseShiftL</a>	Рассчитывает результат бинарной операции SHL для элементов массива.
<a href="#">MathBitwiseShiftR</a>	Рассчитывает результат бинарной операции SHR для элементов массива.
<a href="#">MathCumulativeSum</a>	Формирует массив с накопленной суммой.
<a href="#">MathCumulativeProduct</a>	Формирует массив с накопленным произведением.
<a href="#">MathCumulativeMin</a>	Формирует массив с накопленными минимальными значениями.

<a href="#"><u>MathCumulativeMax</u></a>	Формирует массив с накопленными максимальными значениями.
<a href="#"><u>MathSin</u></a>	Рассчитывает значение функции $\sin(x)$ для элементов массива.
<a href="#"><u>MathCos</u></a>	Рассчитывает значение функции $\cos(x)$ для элементов массива.
<a href="#"><u>MathTan</u></a>	Рассчитывает значение функции $\tan(x)$ для элементов массива.
<a href="#"><u>MathArcsin</u></a>	Рассчитывает значение функции $\arcsin(x)$ для элементов массива.
<a href="#"><u>MathArccos</u></a>	Рассчитывает значение функции $\arccos(x)$ для элементов массива.
<a href="#"><u>MathArctan</u></a>	Рассчитывает значение функции $\arctan(x)$ для элементов массива.
<a href="#"><u>MathSinPi</u></a>	Рассчитывает значение функции $\sin(\pi * x)$ для элементов массива.
<a href="#"><u>MathCosPi</u></a>	Рассчитывает значение функции $\cos(\pi * x)$ для элементов массива.
<a href="#"><u>MathTanPi</u></a>	Рассчитывает значение функции $\tan(\pi * x)$ для элементов массива.
<a href="#"><u>MathAbs</u></a>	Рассчитывает абсолютное значение элементов массива.
<a href="#"><u>MathCeil</u></a>	Возвращает ближайшее сверху целое числовое значение для элементов массива.
<a href="#"><u>MathFloor</u></a>	Возвращает ближайшее снизу целое числовое значение для элементов массива.
<a href="#"><u>MathSqrt</u></a>	Рассчитывает квадратный корень для элементов массива.
<a href="#"><u>MathExp</u></a>	Рассчитывает значение функции $\exp(x)$ для элементов массива.
<a href="#"><u>MathPow</u></a>	Рассчитывает значение функции $\text{pow}(x, power)$ для элементов массива.
<a href="#"><u>MathLog</u></a>	Рассчитывает значение функции $\log(x)$ для элементов массива.
<a href="#"><u>MathLog2</u></a>	Рассчитывает значение логарифма по основанию 2 для элементов массива.
<a href="#"><u>MathLog10</u></a>	Рассчитывает значение логарифма по основанию 10 для элементов массива.

<a href="#"><u>MathDifference</u></a>	Формирует массив с разностями элементов $y[i]=x[i+lag]-x[i]$ .
<a href="#"><u>MathSample</u></a>	Производит случайную выборку элементов массива.
<a href="#"><u>MathTukeySummary</u></a>	Рассчитывает пятичисловую сводку Тьюки для элементов массива.
<a href="#"><u>MathRange</u></a>	Рассчитывает минимальные и максимальные значения элементов массива.
<a href="#"><u>MathMin</u></a>	Возвращает минимальное значение среди всех элементов массива.
<a href="#"><u>MathMax</u></a>	Возвращает максимальное значение среди всех элементов массива.
<a href="#"><u>MathSum</u></a>	Возвращает сумму элементов массива.
<a href="#"><u>MathProduct</u></a>	Возвращает произведение элементов массива.
<a href="#"><u>MathStandardDeviation</u></a>	Рассчитывает стандартное отклонение элементов массива.
<a href="#"><u>MathAverageDeviation</u></a>	Рассчитывает среднее отклонение элементов массива.
<a href="#"><u>MathMedian</u></a>	Рассчитывает медианное значение элементов массива.
<a href="#"><u>MathMean</u></a>	Рассчитывает среднее значение элементов массива.
<a href="#"><u>MathVariance</u></a>	Рассчитывает дисперсию элементов массива.
<a href="#"><u>MathSkewness</u></a>	Рассчитывает коэффициент асимметрии элементов массива.
<a href="#"><u>MathKurtosis</u></a>	Рассчитывает коэффициент эксцесса элементов массива.
<a href="#"><u>MathLog1p</u></a>	Рассчитывает значение функции $\log(1+x)$ для элементов массива.
<a href="#"><u>MathExpm1</u></a>	Рассчитывает значение функции $\exp(x)-1$ для элементов массива.
<a href="#"><u>MathSinh</u></a>	Рассчитывает значение функции $\sinh(x)$ для элементов массива.
<a href="#"><u>MathCosh</u></a>	Рассчитывает значение функции $\cosh(x)$ для элементов массива.
<a href="#"><u>MathTanh</u></a>	Рассчитывает значение функции $\tanh(x)$ для элементов массива.

<a href="#"><u>MathArcsinh</u></a>	Рассчитывает значение функции $\text{arcsinh}(x)$ для элементов массива.
<a href="#"><u>MathArccosh</u></a>	Рассчитывает значение функции $\text{arccosh}(x)$ для элементов массива.
<a href="#"><u>MathArctanh</u></a>	Рассчитывает значение функции $\text{arctanh}(x)$ для элементов массива.
<a href="#"><u>MathSignif</u></a>	Округляет значение до указанного количества знаков в мантиссе.
<a href="#"><u>MathRank</u></a>	Рассчитывает ранги элементов массива.
<a href="#"><u>MathCorrelationPearson</u></a>	Рассчитывает коэффициент корреляции Пирсона.
<a href="#"><u>MathCorrelationSpearman</u></a>	Рассчитывает коэффициент корреляции Спирмена.
<a href="#"><u>MathCorrelationKendall</u></a>	Рассчитывает коэффициент корреляции Кендалла.
<a href="#"><u>MathQuantile</u></a>	Рассчитывает выборочные квантили, соответствующие указанным вероятностям.
<a href="#"><u>MathProbabilityDensityEmpirical</u></a>	Вычисляет эмпирическую плотность вероятности для случайных значений.
<a href="#"><u>MathCumulativeDistributionEmpirical</u></a>	Вычисляет эмпирическое кумулятивное распределение для случайных значений.

## MathRandomNonZero

Возвращает случайное число в диапазоне от 0.0 до 1.0 с плавающей запятой.

```
double MathRandomNonZero()
```

### Возвращаемое значение

Случайное число в диапазоне от 0.0 до 1.0 с плавающей запятой.

## MathMoments

Рассчитывает первые 4 момента элементов массива: среднее, дисперсию, коэффициент асимметрии, коэффициент эксцесса.

```
bool MathMoments(
    const double& array[],           // массив значений
    double& mean,                   // переменная для среднего
    double& variance,              // переменная для дисперсии
    double& skewness,               // переменная для коэффициента асимметрии
    double& kurtosis,               // переменная для коэффициента эксцесса
    const int start=0,              // начальный индекс
    const int count=WHOLE_ARRAY    // количество элементов
)
```

### Параметры

*array[]*

[in] Массив значений.

*mean*

[out] Переменная для среднего значения (1 момент).

*variance*

[out] Переменная для дисперсии (2 момент).

*skewness*

[out] Переменная для коэффициента асимметрии (3 момент).

*kurtosis*

[out] Переменная для коэффициента эксцесса (4 момент).

*start=0*

[in] Начальный индекс для расчета.

*count=WHOLE\_ARRAY*

[in] Количество элементов для расчета.

### Возвращаемое значение

Возвращает `true`, если моменты успешно рассчитаны, иначе `false`.

## MathPowInt

Возводит число в указанную целочисленную степень.

```
double MathPowInt(  
    const double x,           // значение числа  
    const int    power        // степень возведения  
)
```

### Параметры

*x*

[in] Число двойной точности с плавающей запятой, возводимое в степень.

*power*

[in] Целое число, задающее степень.

### Возвращаемое значение

Число *x*, введенное в указанную степень.

## MathFactorial

Рассчитывает факториал заданного целого числа.

```
double MathFactorial(
    const int n // значение числа
)
```

### Параметры

*n*

[in] Целое число, факториал которого необходимо вычислить.

### Возвращаемое значение

Факториал числа.

## MathTrunc

Рассчитывает целую часть заданного числа или элементов массива.

Версия для работы с числом двойной точности с плавающей запятой:

```
double MathTrunc(
    const double x           // значение числа
)
```

### Возвращаемое значение

Целая часть заданного числа.

Версия для работы с массивом чисел двойной точности с плавающей запятой. Результаты записываются в новый массив:

```
bool MathTrunc(
    const double& array[],   // массив значений
    double&        result[] // массив результатов
)
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

Версия для работы с массивом чисел двойной точности с плавающей запятой. Результаты записываются в этот же массив:

```
bool MathTrunc(
    double&       array[]    // массив значений
)
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Параметры

*x*

[in] Число двойной точности с плавающей запятой, целую часть которого нужно получить.

*array[]*

[in] Массив чисел двойной точности с плавающей запятой, целую часть которых нужно получить.

*array[]*

[out] Массив выходных значений.

*result[]*

[out] Массив выходных значений.

## MathRound

Округляет число двойной точности с плавающей запятой или массив таких чисел до заданного количества дробных разрядов.

Версия для округления числа двойной точности с плавающей запятой до заданного количества дробных разрядов:

```
double MathRound(
    const double x,           // значение числа
    const int    digits       // количество знаков после запятой
)
```

### Возвращаемое значение

Число, ближайшее к параметру *x*, количество цифр дробной части которого равно *digits*.

Версия для округления массива чисел двойной точности с плавающей запятой до заданного количества дробных разрядов. Результаты записываются в новый массив.

```
bool MathRound(
    const double& array[],     // массив значений
    int            digits,      // количество знаков после запятой
    double&        result[]    // массив результатов
)
```

### Возвращаемое значение

Возвращает *true* в случае успеха, иначе *false*.

Версия для округления массива чисел двойной точности с плавающей запятой до заданного количества дробных разрядов. Результаты записываются в этот же массив.

```
bool MathRound(
    double&        array[],     // массив значений
    int             digits,      // количество знаков после запятой
)
```

### Возвращаемое значение

Возвращает *true* в случае успеха, иначе *false*.

## Параметры

*x*

[in] Округляемое число двойной точности с плавающей запятой.

*digits*

[in] Количество цифр дробной части в возвращаемом значении.

*array[]*

[in] Массив округляемых чисел двойной точности с плавающей запятой.

*array[]*

[out] Массив выходных значений.

*result[]*

[out] Массив выходных значений.

## MathArctan2

Возвращает арктангенс от частного двух аргументов ( $x$ ,  $y$ ).

Версия для работы с отношением двух указанных чисел ( $x$ ,  $y$ ):

```
double MathArctan2(
    const double      y,           // координата у
    const double      x            // координата х
)
```

### Возвращаемое значение

Угол,  $\theta$ , измеренный в радианах, такой, что  $-\pi \leq \theta \leq \pi$ , и  $\tan(\theta) = y/x$ , где  $(x, y)$  – это точка в декартовой системе координат.

Версия для работы с отношением пары элементов из массивов  $x$  и  $y$ :

```
bool MathArctan2(
    const double&      x[],        // массив значений x
    const double&      y[],        // массив значений y
    double&            result[]   // массив результатов
)
```

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`.

### Параметры

$y$

[in] Координата  $y$  точки.

$x$

[in] Координата  $x$  точки.

$x[]$

[in] Массив координат  $x$  точек.

$y[]$

[in] Массив координат  $y$  точек.

$result[]$

[out] Массив для записи результатов

### Примечания

Обратите внимание на следующее.

- Для  $(x, y)$  в квадранте 1 возвращаемое значение будет:  $0 < \theta < \pi/2$ .
- Для  $(x, y)$  в квадранте 2 возвращаемое значение будет:  $\pi/2 < \theta \leq \pi$ .
- Для  $(x, y)$  в квадранте 3 возвращаемое значение будет:  $-\pi < \theta < -\pi/2$ .
- Для  $(x, y)$  в квадранте 4 возвращаемое значение будет:  $-\pi/2 < \theta < 0$ .

Для точек за пределами указанных квадрантов возвращаемое значение указано ниже.

- Если у равно 0 и x не является отрицательным,  $\theta = 0$ .
- Если у равно 0 и x является отрицательным,  $\theta = \pi$ .
- Если у – положительное число, а x равно 0,  $\theta = \pi/2$ .
- Если у является отрицательным и x равно 0,  $\theta = -\pi/2$ .
- Если у равен 0 и x равен 0, то  $\theta = -\pi/2$ .

Если значение параметра x или у равно NaN либо если значения параметров x и у равны значению PositiveInfinity или NegativeInfinity, метод возвращает значение NaN.

## MathGamma

Вычисляет значение гамма-функции для вещественного аргумента x.

```
double MathGamma(  
    const double x           // аргумент функции  
)
```

### Параметры

x

[in] Вещественный аргумент функции.

### Возвращаемое значение

Значение гамма-функции.

## MathGammaLog

Вычисляет значение логарифма гамма-функции для вещественного аргумента x.

```
double MathGammaLog(  
    const double x          // аргумент функции  
)
```

### Параметры

x

[in] Вещественный аргумент функции.

### Возвращаемое значение

Значение логарифма функции.

## MathBeta

Вычисляет значение бета-функции для вещественных аргументов a и b.

```
double MathBeta(  
    const double a,           // первый аргумент функции  
    const double b            // второй аргумент функции  
)
```

### Параметры

a

[in] Аргумент функции a.

b

[in] Аргумент функции b.

### Возвращаемое значение

Значение функции.

## MathBetaLog

Вычисляет значение логарифма бета-функции для вещественных аргументов a и b.

```
double MathBetaLog(  
    const double a,           // первый аргумент функции  
    const double b            // второй аргумент функции  
)
```

### Параметры

a

[in] Аргумент функции a.

b

[in] Аргумент функции b.

### Возвращаемое значение

Значение логарифма функции.

## MathBetaIncomplete

Рассчитывает значение неполной бета-функции.

```
double MathBetaIncomplete(
    const double x,           // аргумент функции
    const double p,           // первый параметр функции
    const double q            // второй параметр функции
)
```

### Параметры

*x*

[in] Аргумент функции.

*p*

[in] Первый параметр бета-функции, должен быть >0.0.

*q*

[in] Второй параметр бета-функции, должен быть >0.0.

### Возвращаемое значение

Значение функции.

## MathGammaIncomplete

Вычисляет значение функции неполной гамма-функции.

```
double MathGammaIncomplete(  
    double x,           // аргумент функции  
    double alpha        // параметр функции  
)
```

### Параметры

*x*

[in] Аргумент функции.

*alpha*

[in] Параметр неполной гамма-функции.

### Возвращаемое значение

Значение функции.

## MathBinomialCoefficient

Рассчитывает биномиальный коэффициент:  $C(n,k)=n!/(k!*(n-k)!)$ .

```
long MathBinomialCoefficient(
    const int n,           // общее количество элементов
    const int k             // количество элементов в сочетании
)
```

### Параметры

*n*

[in] Количество элементов.

*k*

[in] Количество элементов для каждого сочетания.

### Возвращаемое значение

Число комбинаций из N по K.

## MathBinomialCoefficientLog

Рассчитывает логарифм биномиального коэффициента:  $\text{Log}(C(n,k))=\text{Log}(n!/(k!*(n-k)!))$

Версия для целочисленных аргументов:

```
double MathBinomialCoefficientLog(
    const int     n,          // общее количество элементов
    const int     k           // количество элементов в сочетании
)
```

Версия для вещественных аргументов:

```
double MathBinomialCoefficientLog(
    const double   n,          // общее количество элементов
    const double   k           // количество элементов в сочетании
)
```

### Параметры

*n*

[in] Количество элементов.

*k*

[in] Количество элементов для каждого сочетания.

### Возвращаемое значение

Логарифм от  $C(n,k)$ .

## MathHypergeometric2F2

Расчитывает значение функции Hypergeometric\_2F2 (a, b, c, d, z), используя метод Тейлора.

```
double MathHypergeometric2F2(
    const double a,           // первый параметр функции
    const double b,           // второй параметр функции
    const double c,           // третий параметр функции
    const double d,           // четвертый параметр функции
    const double z            // пятый параметр функции
)
```

### Параметры

a

[in] Первый параметр функции.

b

[in] Второй параметр функции.

c

[in] Третий параметр функции.

d

[in] Четвертый параметр функции.

z

[in] Пятый параметр функции.

### Возвращаемое значение

Значение функции.

## MathSequence

Формирует последовательность значений на основе данных значений: первый элемент, последний элемент, шаг последовательности.

Версия для работы с вещественными значениями:

```
bool MathSequence(
    const double from,           // начальное значение
    const double to,             // конечное значение
    const double step,           // шаг
    double& result[]            // массив результатов
)
```

Версия для работы с целочисленными значениями:

```
bool MathSequence(
    const int from,              // начальное значение
    const int to,                // конечное значение
    const int step,              // шаг
    int& result[]               // массив результатов
)
```

### Параметры

*from*

[in] Первое значение последовательности

*to*

[in] Последнее значение последовательности

*step*

[in] Шаг последовательности.

*result[]*

[out] Массив для записи последовательности.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathSequenceByCount

Формирует последовательность значений на основе данных значений: первый элемент, последний элемент, количество элементов последовательности.

Версия для работы с вещественными значениями:

```
bool MathSequenceByCount (
    const double from,           // начальное значение
    const double to,             // конечное значение
    const int count,             // количество
    double& result[]            // массив результатов
)
```

Версия для работы с целочисленными значениями:

```
bool MathSequenceByCount (
    const int from,              // начальное значение
    const int to,                // конечное значение
    const int count,              // количество
    int& result[]                // массив результатов
)
```

### Параметры

*from*

[in] Первое значение последовательности.

*to*

[in] Последнее значение последовательности.

*count*

[in] Количество элементов последовательности.

*result[]*

[out] Массив для записи последовательности.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathReplicate

Формирует повторяющуюся последовательность значений.

Версия для работы с вещественными значениями:

```
bool MathReplicate(
    const double& array[],      // массив значений
    const int      count,        // количество повторов
    double&       result[]     // массив результатов
)
```

Версия для работы с целочисленными значениями:

```
bool MathReplicate(
    const int&    array[],      // массив значений
    const int      count,        // количество повторов
    int&         result[]     // массив результатов
)
```

### Параметры

*array[]*

[in] Массив для формирования последовательности.

*count*

[in] Количество повторений массива в последовательности.

*result[]*

[out] Массив для записи последовательности.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathReverse

Формирует массив значений с обратным порядком элементов.

Версия для работы с вещественными значениями с сохранением результатов в новый массив:

```
bool MathReverse(
    const double& array[],           // массив значений
    double&       result[]          // массив результатов
)
```

Версия для работы с целочисленными значениями с сохранением результатов в новый массив:

```
bool MathReverse(
    const int&      array[],        // массив значений
    int&            result[]       // массив результатов
)
```

Версия для работы с вещественными значениями с сохранением результатов в тот же массив.

```
bool MathReverse(
    double&        array[]         // массив значений
)
```

Версия для работы с целочисленными значениями с сохранением результатов в тот же массив.

```
bool MathReverse(
    int&           array[]         // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*array[]*

[out] Выходной массив с обратным порядком значений.

*result[]*

[out] Выходной массив с обратным порядком значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathIdentical

Сравнивает два массива значений и возвращает true, если совпадают все элементы.

Версия для работы с массивами вещественных значений:

```
bool MathIdentical(
    const double& array1[],           // первый массив значений
    const double& array2[]            // второй массив значений
)
```

Версия для работы с массивами целочисленных значений:

```
bool MathIdentical(
    const int&     array1[],          // первый массив значений
    const int&     array2[]           // второй массив значений
)
```

### Параметры

*array1[]*

[in] Первый массив для сравнения.

*array2[]*

[in] Второй массив для сравнения.

### Возвращаемое значение

Возвращает true, если массивы равны, иначе false.

## MathUnique

Формирует массив только с неповторяющимися значениями.

Версия для работы с вещественными значениями:

```
bool MathUnique(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия для работы с целочисленными значениями:

```
bool MathUnique(
    const int& array[], // массив значений
    int& result[] // массив результатов
)
```

### Параметры

*array[]*

[in] Исходный массив.

*result[]*

[out] Массив для записи уникальных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathQuickSortAscending

Функция для одновременной восходящей сортировки массивов array[] и indices[], использующая алгоритм QuickSort.

```
void MathQuickSortAscending(
    double& array[],           // массив значений
    int&     indices[],        // массив индексов
    int      first,             // начальное значение
    int      last               // конечное значение
)
```

### Параметры

*array[]*

[in][out] Массив для сортировки.

*indices[]*

[in][out] Массив для сохранения индексов исходного массива.

*first*

[in] Индекс элемента, с которого нужно начать сортировку.

*last*

[in] Индекс элемента, на котором нужно закончить сортировку.

## MathQuickSortDescending

Функция для одновременной нисходящей сортировки массивов array[] и indices[], использующая алгоритм QuickSort.

```
void MathQuickSortDescending(
    double& array[],           // массив значений
    int&     indices[],        // массив индексов
    int      first,             // начальное значение
    int      last               // конечное значение
)
```

### Параметры

*array[]*

[in][out] Массив для сортировки.

*indices[]*

[in][out] Массив для сохранения индексов исходного массива.

*first*

[in] Индекс элемента, с которого нужно начать сортировку.

*last*

[in] Индекс элемента, на котором нужно закончить сортировку.

## MathQuickSort

Функция для одновременной сортировки массивов `array[]` и `indices[]`, использующая алгоритм QuickSort.

```
void MathQuickSort(
    double& array[],           // массив значений
    int&     indices[],        // массив индексов
    int      first,             // начальное значение
    int      last,              // конечное значение
    int      mode               // направление
)
```

### Параметры

*array[]*

[in][out] Массив для сортировки.

*indices[]*

[in][out] Массив для сохранения индексов исходного массива.

*first*

[in] Индекс элемента, с которого нужно начать сортировку.

*last*

[in] Индекс элемента, на котором нужно закончить сортировку.

*mode*

[in] Направление сортировки (>0 восходящая, иначе нисходящая).

## MathOrder

Формирует целочисленный массив с перестановкой в соответствии с порядком элементов массива после сортировки.

Версия для работы с массивом вещественных значений:

```
bool MathOrder(
    const double& array[], // массив значений
    int& result[] // массив результатов
)
```

Версия для работы с массивом целочисленных значений:

```
bool MathOrder(
    const int& array[], // массив значений
    int& result[] // массив результатов
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив для записи отсортированных индексов.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathBitwiseNot

Рассчитывает результат бинарной операции NOT для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathBitwiseNot(
    const int& array[], // массив значений
    int&         result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathBitwiseNot(
    int&         array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*array[]*

[out] Массив выходных значений.

*result[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathBitwiseAnd

Рассчитывает результат бинарной операции AND для заданных массивов.

```
bool MathBitwiseAnd(
    const int& array1[], // первый массив значений
    const int& array2[], // второй массив значений
    int&         result[] // массив результатов
)
```

### Параметры

*array1[]*

[in] Первый массив значений.

*array2[]*

[in] Второй массив значений.

*result[]*

[out] Массив для записи результатов.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathBitwiseOr

Рассчитывает результат бинарной операции OR для заданных массивов.

```
bool MathBitwiseOr(
    const int& array1[], // первый массив значений
    const int& array2[], // второй массив значений
    int&         result[] // массив результатов
)
```

### Параметры

*array1[]*

[in] Первый массив значений.

*array2[]*

[in] Второй массив значений.

*result[]*

[out] Массив для записи результатов.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathBitwiseXor

Рассчитывает результат бинарной операции XOR для заданных массивов.

```
bool MathBitwiseXor(
    const int& array1[], // первый массив значений
    const int& array2[], // второй массив значений
    int&         result[] // массив результатов
)
```

### Параметры

*array1[]*

[in] Первый массив значений.

*array2[]*

[in] Второй массив значений.

*result[]*

[out] Массив для записи результатов.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathBitwiseShiftL

Рассчитывает результат бинарной операции SHL (побитовый сдвиг влево) для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathBitwiseShiftL(
    const int& array[],           // массив значений
    const int    n,                // значение сдвига
    int&        result[]          // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathBitwiseShiftL(
    int&       array[],           // массив значений
    const int    n                 // значение сдвига
)
```

### Параметры

*array[]*

[in] Массив значений.

*n*

[in] Число битов для сдвига.

*array[]*

[out] Массив выходных значений.

*result[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathBitwiseShiftR

Рассчитывает результат бинарной операции SHR (побитовый сдвиг вправо) для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathBitwiseShiftR(
    const int& array[],      // массив значений
    const int    n,           // значение сдвига
    int&        result[]    // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathBitwiseShiftR(
    int&       array[],      // массив значений
    const int    n            // значение сдвига
)
```

### Параметры

*array[]*

[in] Массив значений.

*n*

[in] Число битов для сдвига.

*array[]*

[out] Массив выходных значений.

*result[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCumulativeSum

Формирует массив с накопленной суммой.

Версия с записью результатов в новый массив:

```
bool MathCumulativeSum(
    const double& array[],      // массив значений
    double&       result[]      // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathCumulativeSum(
    double&       array[]       // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*array[]*

[out] Массив выходных значений.

*result[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCumulativeProduct

Формирует массив с накопленным произведением.

Версия с записью результатов в новый массив:

```
bool MathCumulativeProduct(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathCumulativeProduct(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCumulativeMin

Формирует массив с накопленными минимальными значениями.

Версия с записью результатов в новый массив:

```
bool MathCumulativeMin(
    const double& array[],      // массив значений
    double&       result[]       // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathCumulativeMin(
    double&       array[]        // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCumulativeMax

Формирует массив с накопленными максимальными значениями.

Версия с записью результатов в новый массив:

```
bool MathCumulativeMax(
    const double& array[],      // массив значений
    double&       result[]      // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathCumulativeMax(
    double&       array[]      // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathSin

Рассчитывает значение функции  $\sin(x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathSin(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathSin(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCos

Рассчитывает значение функции cos(x) для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathCos(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathCos(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathTan

Рассчитывает значение функции  $\tan(x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathTan(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathTan(
    double& array[] // массив значений
)
```

### Параметры

*array[]*  
[in] Массив значений.

*result[]*  
[out] Массив выходных значений.

*array[]*  
[out] Массив выходных значений.

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`.

## MathArcsin

Рассчитывает значение функции  $\arcsin(x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathArcsin(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathArcsin(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`.

## MathArccos

Рассчитывает значение функции arccos(x) для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathArccos(
    const double& array[],      // массив значений
    double&       result[]      // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathArccos(
    double&       array[]      // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathArctan

Рассчитывает значение функции arctan(x) для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathArctan(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathArctan(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathSinPi

Рассчитывает значение функции  $\sin(pi*x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathSinPi(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathSinPi(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCosPi

Рассчитывает значение функции  $\cos(pi*x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathCosPi(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathCosPi(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathTanPi

Рассчитывает значение функции  $\tan(\pi \cdot x)$  для элементов массива.

Версия с записью результата в новый массив:

```
bool MathTanPi(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результата в исходный массив:

```
bool MathTanPi(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathAbs

Рассчитывает абсолютное значение элементов массива.

Версия с записью результатов в новый массив:

```
bool MathAbs(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathAbs(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCeil

Возвращает ближайшее сверху целое числовое значение для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathCeil(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathCeil(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathFloor

Возвращает ближайшее снизу целое числовое значение для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathFloor(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathFloor(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathSqrt

Рассчитывает квадратный корень для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathSqrt(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathSqrt(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathExp

Рассчитывает значение функции  $\exp(x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathExp(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathExp(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`.

## MathPow

Рассчитывает значение функции pow(x, power) для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathPow(
    const double& array[],      // массив значений
    const double    power,        // степень
    double&        result[]     // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathPow(
    double&       array[],      // массив значений
    const double   power         // степень
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathLog

Рассчитывает значение функции  $\log(x)$  для элементов массива.

Версия для расчета натурального логарифма с записью результатов в новый массив.

```
bool MathLog(
    const double& array[],           // массив значений
    double&       result[]          // массив результатов
)
```

Версия для расчета натурального логарифма с записью результатов в исходный массив.

```
bool MathLog(
    double&      array[]           // массив значений
)
```

Версия для расчета логарифма по заданному основанию с записью результатов в новый массив.

```
bool MathLog(
    const double& array[],         // массив значений
    const double    base,            // основание логарифма
    double&       result[]          // массив результатов
)
```

Версия для расчета логарифма по заданному основанию с записью результатов в исходный массив.

```
bool MathLog(
    double&      array[],          // массив значений
    const double   base             // основание логарифма
)
```

### Параметры

*array[]*

[in] Массив значений.

*base*

[in] Основание логарифма.

*array[]*

[out] Массив выходных значений.

*result[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathLog2

Рассчитывает значение логарифма по основанию 2 для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathLog2(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathLog2(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathLog10

Рассчитывает значение логарифма по основанию 10 для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathLog10(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathLog10(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathLog1p

Рассчитывает значение функции  $\log(1+x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathLog1p(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathLog1p(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathDifference

Формирует массив с разностями элементов  $y[i] = x[i+lag] - x[i]$ .

Версия для однократного формирования массива вещественных значений:

```
bool MathDifference(
    const double &array[],           // массив значений
    const int     lag,                // запаздывание
    double       &result[]            // массив результатов
)
```

Версия для однократного формирования массива целочисленных значений:

```
bool MathDifference(
    const int     &array[],           // массив значений
    const int     lag,                // запаздывание
    int          &result[]            // массив результатов
)
```

Версия для многократного формирования массива вещественных значений (количество итераций задается во входных параметрах):

```
bool MathDifference(
    const double &array[],           // массив значений
    const int     lag,                // запаздывание
    const int     differences,        // количество итераций
    double       &result[]            // массив результатов
)
```

Версия для многократного формирования массива целочисленных значений (количество итераций задается во входных параметрах):

```
bool MathDifference(
    const int&   array[],           // массив значений
    const int     lag,                // запаздывание
    const int     differences,        // количество итераций
    int&         result[]           // массив результатов
)
```

### Параметры

*array[]*

[in] Массив значений.

*lag*

[in] Параметр запаздывания.

*differences*

[in] Количество итераций.

*result[]*

[out] Массив для записи результатов.

#### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathSample

Производит случайную выборку элементов массива.

Версия для работы с массивом вещественных значений:

```
bool MathSample(
    const double& array[],           // массив значений
    const int      count,             // количество
    double&       result[]          // массив результатов
)
```

Версия для работы с массивом целочисленных значений:

```
bool MathSample(
    const int&     array[],           // массив значений
    const int      count,             // количество
    int&          result[]          // массив результатов
)
```

Версия для работы с массивом вещественных значений. Присутствует возможность получить выборку с возвратом:

```
bool MathSample(
    const double& array[],           // массив значений
    const int      count,             // количество
    const bool     replace,            // флаг
    double&       result[]          // массив результатов
)
```

Версия для работы с массивом целочисленных значений. Присутствует возможность получить выборку с возвратом:

```
bool MathSample(
    const int&     array[],           // массив значений
    const int      count,             // количество
    const bool     replace,            // флаг
    int&          result[]          // массив результатов
)
```

Версия для работы с массивом вещественных значений, для которых определены вероятности попадания в выборку.

```
bool MathSample(
    const double& array[],           // массив значений
    double&       probabilities[],   // массив вероятностей
    const int      count,             // количество
    double&       result[]          // массив результатов
)
```

Версия для работы с массивом целочисленных значений, для которых определены вероятности попадания в выборку.

```
bool MathSample(
    const int& array[],           // массив значений
    double& probabilities[],     // массив вероятностей
    const int count,              // количество
    int& result[]                // массив результатов
)
```

Версия для работы с массивом вещественных значений, для которых определены вероятности попадания в выборку. Есть возможность получения выборки с возвратом:

```
bool MathSample(
    const double& array[],         // массив значений
    double& probabilities[],       // массив вероятностей
    const int count,               // количество
    const bool replace,            // флаг
    double& result[]              // массив результатов
)
```

Версия для работы с массивом целочисленных значений, для которых определены вероятности попадания в выборку. Есть возможность получения выборки с возвратом:

```
bool MathSample(
    const int& array[],           // массив значений
    double& probabilities[],     // массив вероятностей
    const int count,              // количество
    const bool replace,            // флаг
    int& result[]                // массив результатов
)
```

## Параметры

*array[]*

[in] Массив целочисленных значений.

*probabilities[]*

[in] Массив вероятностей, с которыми производится выборка элементов.

*count*

[in] Количество элементов.

*replace*

[in] Параметр, позволяющий осуществлять отбор с возвратом.

*result[]*

[out] Массив для записи результатов.

## Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Примечание

Аргумент `replace=true` позволяет осуществлять случайный отбор элементов с их возвратом в исходную совокупность.

## MathTukeySummary

Рассчитывает пятичисловую сводку Тьюки (минимум, нижний квартиль, среднее, верхний квартиль, максимум) для элементов массива.

```
bool MathTukeySummary(
    const double& array[],           // массив значений
    const bool      removeNAN,        // флаг
    double&        minimum,          // минимальное значение
    double&        lower_hinge,       // нижний квартиль
    double&        median,           // среднее значение
    double&        upper_hinge,       // верхний квартиль
    double&        maximum          // максимальное значение
)
```

### Параметры

*array[]*

[in] Массив вещественных значений.

*removeNAN*

[in] Флаг, указывающий, нужно ли удалять нечисловые значения.

*minimum*

[out] Переменная для записи минимального значения.

*lower\_hinge*

[out] Переменная для записи нижнего квартиля.

*median*

[out] Переменная для записи среднего значения.

*upper\_hinge*

[out] Переменная для записи верхнего квартиля.

*maximum*

[out] Переменная для записи максимального значения.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathRange

Рассчитывает минимальные и максимальные значения элементов массива.

```
bool MathRange(
    const double& array[], // массив значений
    double& min,          // минимальное значение
    double& max           // максимальное значение
)
```

### Параметры

*array[]*

[in] Массив значений.

*min*

[out] Переменная для записи минимального значения.

*max*

[out] Переменная для записи максимального значения.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathMin

Возвращает минимальное значение среди всех элементов массива.

```
double MathMin(
    const double& array[]    // массив значений
)
```

### Параметры

*array[]*  
[in] Массив значений.

### Возвращаемое значение

Минимальное значение.

## MathMax

Возвращает максимальное значение среди всех элементов массива.

```
double MathMax(  
    const double& array[] // массив значений  
)
```

### Параметры

*array[]*  
[in] Массив значений.

### Возвращаемое значение

Максимальное значение.

## MathSum

Возвращает сумму элементов массива.

```
double MathSum(  
    const double& array[] // массив значений  
)
```

### Параметры

*array[]*  
[in] Массив значений.

### Возвращаемое значение

Сумма элементов.

## MathProduct

Возвращает произведение элементов массива.

```
double MathProduct(
    const double& array[]    // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

### Возвращаемое значение

Произведение элементов.

## MathStandardDeviation

Рассчитывает стандартное отклонение элементов массива.

```
double MathStandardDeviation(
    const double& array[] // массив значений
)
```

### Параметры

*array[]*  
[in] Массив значений.

### Возвращаемое значение

Стандартное отклонение.

## MathAverageDeviation

Функция рассчитывает среднее отклонение элементов массива.

```
double MathAverageDeviation(
    const double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

### Возвращаемое значение

Среднее отклонение элементов массива.

## MathMedian

Рассчитывает медианное значение элементов массива.

```
double MathMedian(  
    double& array[] // массив значений  
)
```

### Параметры

*array[]*  
[in] Массив значений.

### Возвращаемое значение

Медианное значение.

## MathMean

Рассчитывает среднее значение элементов массива.

```
double MathMean(  
    const double& array[] // массив значений  
)
```

### Параметры

*array[]*  
[in] Массив значений.

### Возвращаемое значение

Среднее значение.

## MathVariance

Функция рассчитывает дисперсию (второй момент) элементов массива.

```
double MathVariance(
    const double& array[]    // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

### Возвращаемое значение

Значение дисперсии.

## MathSkewness

Функция рассчитывает коэффициент асимметрии (третий момент) элементов массива.

```
double MathSkewness(
    const double& array[]    // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

### Возвращаемое значение

Коэффициент асимметрии.

## MathKurtosis

Функция рассчитывает коэффициент эксцесса (четвертый момент) элементов массива.

```
double MathKurtosis(
    const double& array[]      // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

### Возвращаемое значение

Коэффициент эксцесса.

## MathExprm1

Рассчитывает значение функции  $\exp(x)-1$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathExprm1(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathExprm1(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathSinh

Рассчитывает значение функции  $\sinh(x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathSinh(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathSinh(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCosh

Рассчитывает значение функции  $\cosh(x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathCosh(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathCosh(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathTanh

Рассчитывает значение функции  $\tanh(x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathTanh(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathTanh(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathArcsinh

Рассчитывает значение функции  $\text{arcsinh}(x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathArcsinh(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathArcsinh(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathArccosh

Рассчитывает значение функции  $\text{arccosh}(x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathArccosh (
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathArccosh (
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`.

## MathArctanh

Рассчитывает значение функции  $\operatorname{arctanh}(x)$  для элементов массива.

Версия с записью результатов в новый массив:

```
bool MathArctanh(
    const double& array[], // массив значений
    double& result[] // массив результатов
)
```

Версия с записью результатов в исходный массив:

```
bool MathArctanh(
    double& array[] // массив значений
)
```

### Параметры

*array[]*

[in] Массив значений.

*result[]*

[out] Массив выходных значений.

*array[]*

[out] Массив выходных значений.

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`.

## MathSignif

Округляет значение до указанного количества знаков в мантиссе.

Версия для работы с вещественным значением:

```
double MathSignif(
    const double x,           // значение
    const int    digits       // количество знаков
)
```

### Возвращаемое значение

Округленное значение.

Версия для работы с массивом вещественных значений с записью результатов в отдельный массив:

```
bool MathSignif(
    const double& array[],   // массив значений
    int          digits,     // количество знаков
    double       result[]    // массив результатов
)
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

Версия для работы с массивом вещественных значений с записью результатов в исходный массив:

```
bool MathSignif(
    double&      array[],   // массив значений
    int          digits,     // количество знаков
)
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Параметры

*x*

[in] Вещественное значение для округления.

*digits*

[in] Количество знаков.

*array[]*

[in] Массив вещественных значений.

*array[]*

[out] Массив выходных значений.

*result[]*

[out] Массив выходных значений.



## MathRank

Рассчитывает ранги элементов массива.

Версия для работы с массивом вещественных значений:

```
bool MathRank(
    const double& array[], // массив значений
    double& rank[]        // массив рангов
)
```

Версия для работы с массивом целочисленных значений:

```
bool MathRank(
    const int& array[], // массив значений
    double& rank[]      // массив рангов
)
```

### Параметры

*array[]*

[in] Массив значений.

*rank[]*

[out] Массив для записи рангов.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCorrelationPearson

Рассчитывает коэффициент корреляции Пирсона.

Версия для работы с массивами вещественных значений:

```
bool MathCorrelationPearson(
    const double& array1[], // первый массив значений
    const double& array2[], // второй массив значений
    double& r             // коэффициент корреляции
)
```

Версия для работы с массивами целочисленных значений:

```
bool MathCorrelationPearson(
    const int& array1[], // первый массив значений
    const int& array2[], // второй массив значений
    double& r           // коэффициент корреляции
)
```

### Параметры

*array1[]*

[in] Первый массив значений.

*array2[]*

[in] Второй массив значений.

*r*

[out] Переменная для записи коэффициента корреляции.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCorrelationSpearman

Рассчитывает коэффициент корреляции Спирмена.

Версия для работы с массивами вещественных значений:

```
bool MathCorrelationSpearman(
    const double& array1[], // первый массив значений
    const double& array2[], // второй массив значений
    double& r             // коэффициент корреляции
)
```

Версия для работы с массивами целочисленных значений:

```
bool MathCorrelationSpearman(
    const int& array1[], // первый массив значений
    const int& array2[], // второй массив значений
    double& r           // коэффициент корреляции
)
```

### Параметры

*array1[]*

[in] Первый массив значений.

*array2[]*

[in] Второй массив значений.

*r*

[out] Переменная для записи коэффициента корреляции.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCorrelationKendall

Рассчитывает коэффициент корреляции Кендалла.

Версия для работы с массивами вещественных значений:

```
bool MathCorrelationKendall(
    const double& array1[], // первый массив значений
    const double& array2[], // второй массив значений
    double& tau           // коэффициент корреляции
)
```

Версия для работы с массивами целочисленных значений:

```
bool MathCorrelationKendall(
    const int& array1[], // первый массив значений
    const int& array2[], // второй массив значений
    double& tau         // коэффициент корреляции
)
```

### Параметры

*array1[]*

[in] Первый массив значений.

*array2[]*

[in] Второй массив значений.

*tau*

[out] Переменная для записи коэффициента корреляции.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathQuantile

Рассчитывает выборочные квантили, соответствующие указанным вероятностям: Q[i] ( $p$ ) =  $(1 - \text{gamma}) * x[j] + \text{gamma} * x[j+1]$

```
bool MathQuantile(
    const double& array[],           // массив значений
    const double& probs[],           // массив вероятностей
    double&        quantile[]        // массив для записи квантилей
)
```

### Параметры

*array* []

[in] Массив значений.

*probs* []

[in] Массив вероятностей.

*quantile* []

[out] Массив для записи квантилей.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathProbabilityDensityEmpirical

Функция вычисляет эмпирическую плотность вероятности (pdf) для случайных значений из массива.

```
bool MathProbabilityDensityEmpirical(
    const double& array[], // массив случайных значений
    const int      count,  // количество пар
    double&       x[],   // массив значений x
    double&       pdf[]  // массив значений pdf
)
```

### Параметры

*array[]*

[in] Массив случайных значений.

*count*

[in] Количество пар ( $x, \text{pdf}(x)$ ).

*x[]*

[out] Массив для записи значений  $x$ .

*pdf[]*

[out] Массив для записи значений  $\text{pdf}(x)$ .

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## MathCumulativeDistributionEmpirical

Функция вычисляет эмпирическое кумулятивное распределение(cdf) для случайных значений из массива.

```
bool MathCumulativeDistributionEmpirical(
    const double& array[],           // массив случайных значений
    const int      count,             // количество пар
    double&       x[],               // массив значений x
    double&       cdf[]              // массив значений cdf
)
```

### Параметры

*array[]*

[in] Массив случайных значений.

*count*

[in] Количество пар ( $x, cdf(x)$ ).

*x[]*

[out] Массив для записи значений  $x$ .

*cdf[]*

[out] Массив для записи значений  $cdf(x)$ .

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`.

## Fuzzy – библиотека для работы с нечеткой логикой

**Нечеткая логика** – это обобщение традиционной аристотелевой логики на случай, когда истинность рассматривается как лингвистическая переменная. Как и в классической логике, для нечеткой логики определены свои нечеткие логические операции над нечеткими множествами. Для нечетких множеств существуют все те же операции, что и для обычных множеств, только их вычисление на порядок сложнее. Отметим также, что композиция нечетких множеств – есть нечеткое множество.

Основными особенностями нечеткой логики, отличающими ее от классической, являются максимальная приближенность к отражению реальности и высокий уровень субъективности, вследствие чего могут возникнуть значительные погрешности в результатах вычислений с ее использованием.

Нечеткая модель (или система) – математическая модель, в основе вычисления которой лежит нечеткая логика. К построению таких моделей прибегают в случае, когда предмет исследования имеет очень слабую формализацию, и его точное математическое описание слишком сложное или неизвестно. Качество выходных значений этих моделей (погрешность модели) напрямую зависит только от эксперта, который составлял и настраивал модель. Для минимизации ошибки лучшим вариантом будет составление максимально полной и исчерпывающей модели и последующая ее настройка средствами машинного обучения на большой обучающей выборке.

Ход построения модели можно разделить на три основных этапа:

1. Определение входных и выходных параметров модели.
2. Построение базы знаний.
3. Выбор одного из методов нечеткого логического вывода ([Мамдани](#) или [Сугено](#)).

От первого этапа непосредственно зависят два других, и именно он определяет будущее функционирование модели. База знаний или, как по-другому ее называют, база правил – это совокупность нечетких правил вида: "если, то", определяющих взаимосвязь между входами и выходами исследуемого объекта. Количество правил в системе не ограничено и также определяется экспертом. Обобщенный формат нечетких правил такой:

Если условие (посылка) правила, то заключение правила.

Условие правила характеризует текущее состояние объекта, а заключение – то, как это условие повлияет на объект. Общий вид условий и заключений нельзя выделить, так как они определяются нечетким логическим выводом.

Каждое правило в системе имеет вес – данный параметр характеризует значимость правила в модели. Весовые коэффициенты присваиваются правилу в диапазоне [0, 1]. Во многих примерах нечетких моделей, которые можно встретить в литературе, данные веса не указаны, но это не означает, что их нет, в действительности для каждого правила из базы в таком случае вес фиксирован и равен единице. Условия и заключения для каждого правила могут быть двух видов:

1. простое – в нем участвует одна нечеткая переменная;
2. составное – участвуют несколько нечетких переменных.

В зависимости от созданной базы знаний для модели определяется система нечеткого логического вывода. Нечетким логическим выводом называется получение заключения в виде нечеткого множества, соответствующего текущим значениям входов, с использованием нечеткой базы

знаний и нечетких операций. Два основных типа нечеткого логического вывода – Мамдани и Сугено.

## ФУНКЦИИ ПРИНАДЛЕЖНОСТИ

Функцией принадлежности (*membership function*) называется функция, которая позволяет вычислить степень принадлежности произвольного элемента универсального множества нечеткому множеству. Следовательно, область значений функции принадлежности должна принадлежать диапазону [0, 1].

В большинстве случаев функция принадлежности монотонна и непрерывна.

Класс функций принадлежности	Описание
<a href="#">CConstantMembershipFunction</a>	Класс для реализации функции принадлежности в виде прямой, параллельной оси координат
<a href="#">CCompositeMembershipFunction</a>	Класс для реализации композиции функций принадлежности
<a href="#">CDifferencTwoSigmoidalMembershipFunction</a>	Класс для реализации функции принадлежности в виде разности между двумя сигмоидными функциями с параметрами A1, A2, C1, C2
<a href="#">CGeneralizedBellShapedMembershipFunction</a>	Класс для реализации обобщенной колоколообразной функции принадлежности с параметрами A, B и C
<a href="#">CNormalCombinationMembershipFunction</a>	Класс для реализации двухсторонней гауссовой функции принадлежности с параметрами B1, B2, Sigma1 и Sigma2
<a href="#">CNormalMembershipFunction</a>	Класс для реализации симметричной гауссовой функции принадлежности с параметрами B и Sigma
<a href="#">CP_ShapedMembershipFunction</a>	Класс для реализации пи-подобной функции принадлежности с параметрами A, B, C и D
<a href="#">CProductTwoSigmoidalMembershipFunction</a>	Класс для реализации функции принадлежности в виде произведения двух сигмоидных функций с параметрами A1, A2, C1, C2
<a href="#">CS_ShapedMembershipFunction</a>	Класс для реализации S-подобной функции принадлежности с параметрами A и B
<a href="#">CTrapezoidMembershipFunction</a>	Класс для реализации трапециевидной функции принадлежности с параметрами X1, X2, X3 и X4
<a href="#">CTriangularMembershipFunction</a>	Класс для реализации треугольной функции принадлежности с параметрами X1, X2, X3
<a href="#">CSigmoidalMembershipFunction</a>	Класс для реализации сигмоидной функции принадлежности с параметрами A и C

<a href="#"><u>CZ_ShapedMembershipFunction</u></a>	Класс для реализации z-подобной функции принадлежности с параметрами А и В.
<a href="#"><u>IMembershipFunction</u></a>	Базовый класс для всех классов функций принадлежности.

## CConstantMembershipFunction

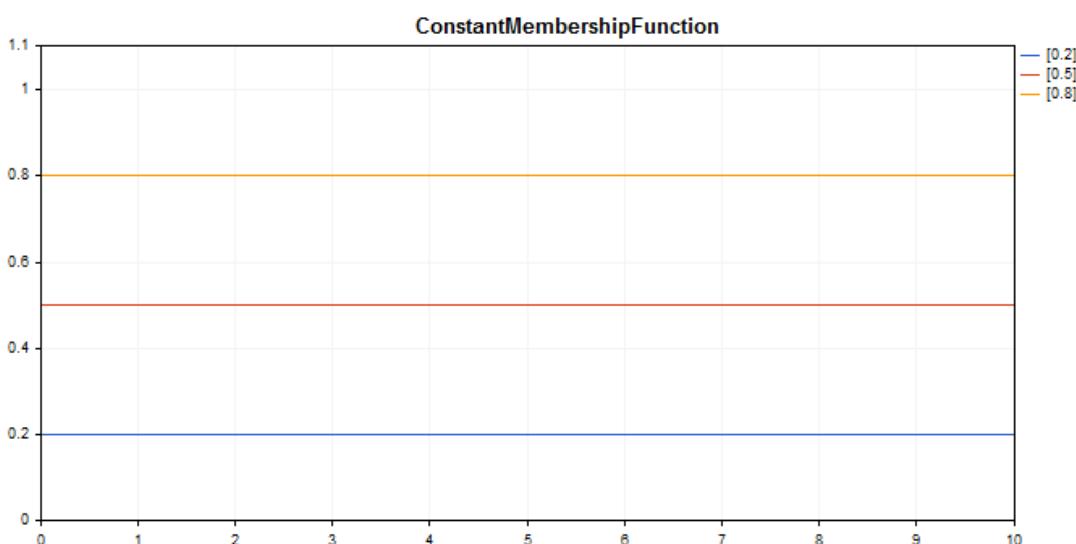
Класс для реализации функции принадлежности в виде прямой, параллельной оси координат.

### Описание

Функция описывается уравнением:

$$y(x) = c$$

Следовательно, степень принадлежности для данной функции одинакова на всей числовой оси и равняется параметру, указанному в конструкторе.



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CConstantMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject
IMembershipFunction
CConstantMembershipFunction
```

### Методы класса

Метод класса	Описание
<a href="#">GetValue</a>	Рассчитывает значение функции принадлежности по указанному аргументу.

**Методы унаследованные от CObject**
[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)
**Пример**

```

//+-----+
//|                                         ConstantMembershipFunction.mq5 |
//|                                         Copyright 2016, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CConstantMembershipFunction func1(0.2);
CConstantMembershipFunction func2(0.5);
CConstantMembershipFunction func3(0.8);
//--- Create wrappers for membership functions
double ConstantMembershipFunction1(double x) { return(func1.GetValue(x)); }
double ConstantMembershipFunction2(double x) { return(func2.GetValue(x)); }
double ConstantMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0, "ConstantMembershipFunction", 0, 30, 30, 780, 380))
{
    graphic.Attach(0, "ConstantMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("ConstantMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(ConstantMembershipFunction1,0.0,10.0,1.0,CURVE_LINES,"[0.2]");
graphic.CurveAdd(ConstantMembershipFunction2,0.0,10.0,1.0,CURVE_LINES,"[0.5]");
graphic.CurveAdd(ConstantMembershipFunction3,0.0,10.0,1.0,CURVE_LINES,"[0.8]");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot

```

```
graphic.CurvePlotAll();
graphic.Update();
}
```

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(
    const double x          // аргумент функции принадлежности
)
```

### Параметры

*x*

[in] Аргумент функции принадлежности.

### Возвращаемое значение

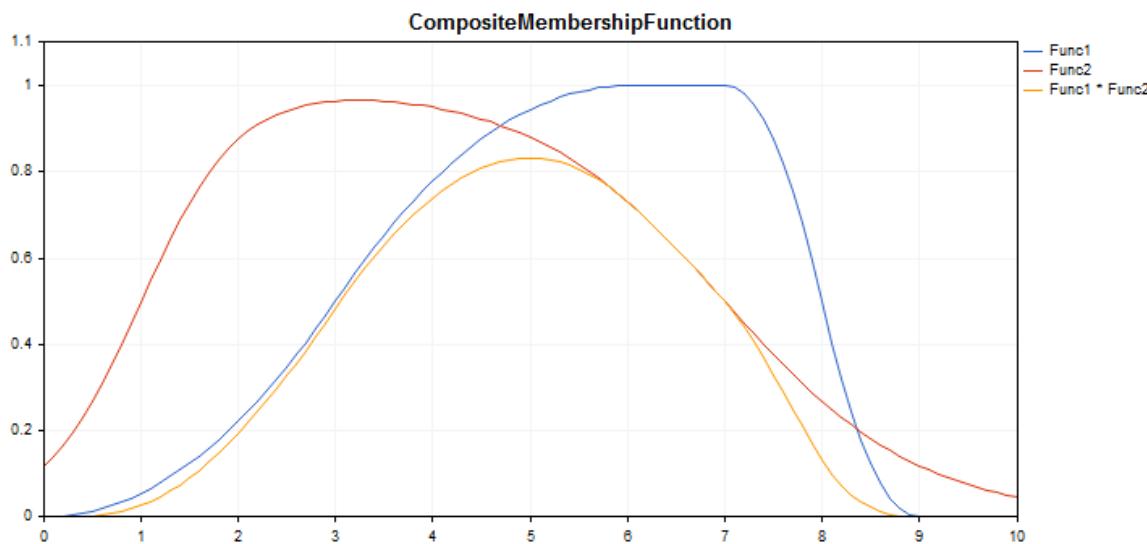
Значение функции принадлежности.

## CCompositeMembershipFunction

Класс для реализации композиции функций принадлежности.

### Описание

Композиция функций принадлежности – объединение двух или более функций принадлежности с помощью заданного оператора.



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CCompositeMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject
IMembershipFunction
CCompositeMembershipFunction
```

### Методы класса

Метод класса	Описание
<a href="#">CompositionType</a>	Устанавливает оператор композиции.
<a href="#">MembershipFunctions</a>	Возвращает список функций принадлежности.
<a href="#">GetValue</a>	Рассчитывает значение функции принадлежности по указанному аргументу.

**Методы унаследованные от CObject**
[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)
**Пример**

```

//+-----+
//|                                         CompositeMembershipFunction.mq5 |
//|                                         Copyright 2016, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CProductTwoSigmoidalMembershipFunctions func1(2,1,-1,7);
CP_ShapedMembershipFunction func2(0,6,7,9);
CCCompositeMembershipFunction composite(ProdMF,GetPointer(func1),GetPointer(func2));
//--- Create wrappers for membership functions
double ProductTwoSigmoidalMembershipFunctions(double x) { return(func1.GetValue(x)); }
double P_ShapedMembershipFunction(double x) { return(func2.GetValue(x)); }
double CompositeMembershipFunction(double x) { return(composite.GetValue(x)); }
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"CompositeMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"CompositeMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("CompositeMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(P_ShapedMembershipFunction,0.0,10.0,0.1,CURVE_LINES,"Func1");
graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions,0.0,10.0,0.1,CURVE_LINES,"I");
graphic.CurveAdd(CompositeMembershipFunction,0.0,10.0,0.1,CURVE_LINES,"Func1 * Func2");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot

```

```
graphic.CurvePlotAll();
graphic.Update();
}
```

## CompositionType

Устанавливает оператор композиции.

```
void CompositionType(
    MfCompositionType value      // тип оператора
)
```

### Параметры

*value*  
 [in] Тип оператора композиции.

### Примечание

Доступны следующие типы операторов:

- MinMF (минимум функций)
- MaxMF (максимум функций)
- ProdMF (произведение функций)
- SumMF (сумма функций)

## MembershipFunctions

Возвращает список функций принадлежности, входящих в композицию.

```
CList* MembershipFunctions(
    void          // список функций принадлежности
)
```

### Возвращаемое значение

Список функций принадлежности.

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(
    const x      // аргумент функции принадлежности
)
```

### Параметры

*x*  
 [in] Аргумент функции принадлежности.

### Возвращаемое значение

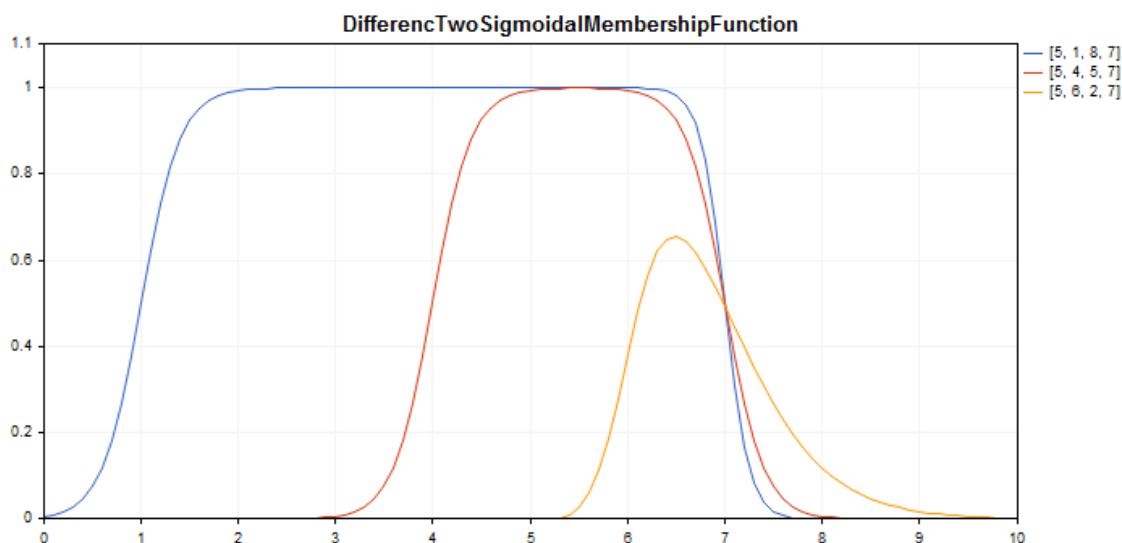
Значение функции принадлежности.

## CDifferencTwoSigmoidalMembershipFunction

Класс для реализации функции принадлежности в виде разности между двумя сигмоидными функциями с параметрами A1, A2, C1, C2.

### Описание

Функция основана на использовании сигмоидной кривой. С ее помощью можно создавать функции принадлежности со значениями, равными 1, начиная с некоторого значения аргумента. Подобные функции подходят, если необходимо задать лингвистические термы типа "короткий" или "длинный".



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CDifferencTwoSigmoidalMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject  
IMembershipFunction  
CDifferencTwoSigmoidalMembershipFunction
```

### Методы класса

Метод класса	Описание
<a href="#">A1</a>	Возвращает и устанавливает коэффициент крутизны первой функции принадлежности.

<u>A2</u>	Возвращает и устанавливает коэффициент крутизны второй функции принадлежности.
<u>C1</u>	Возвращает и устанавливает параметр координаты перегиба первой функции принадлежности.
<u>C2</u>	Возвращает и устанавливает параметр координаты перегиба второй функции принадлежности.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Пример

```

//+-----+
//|          DifferencTwoSigmoidalMembershipFunction.mq5 |
//|          Copyright 2016, MetaQuotes Software Corp. |
//|          https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CDifferencTwoSigmoidalMembershipFunction func1(5,1,8,7);
CDifferencTwoSigmoidalMembershipFunction func2(5,4,5,7);
CDifferencTwoSigmoidalMembershipFunction func3(5,6,2,7);
//--- Create wrappers for membership functions
double DifferencTwoSigmoidalMembershipFunction1(double x) { return(func1.GetValue(x)); }
double DifferencTwoSigmoidalMembershipFunction2(double x) { return(func2.GetValue(x)); }
double DifferencTwoSigmoidalMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"DifferencTwoSigmoidalMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"DifferencTwoSigmoidalMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("DifferencTwoSigmoidalMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,

```

```

graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,
graphic.CurveAdd(DifferencTwoSigmoidalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}

```

## A1 (Метод Get)

Возвращает коэффициент крутизны первой функции принадлежности.

```
double A1()
```

### Возвращаемое значение

Значение коэффициента крутизны.

## A1 (Метод Set)

Устанавливает коэффициент крутизны первой функции принадлежности.

```

void A1(
    const double a1          // значение коэффициента крутизны
)

```

### Параметры

*a1*

[in] Значение коэффициента крутизны.

## A2 (Метод Get)

Возвращает коэффициент крутизны второй функции принадлежности.

```
double A2()
```

### Возвращаемое значение

Значение коэффициента крутизны.

## A2 (Метод Set)

Устанавливает коэффициент крутизны второй функции принадлежности.

```
void A2(
    const double a2          // значение коэффициента крутизны
)
```

#### Параметры

*a2*

[in] Значение коэффициента крутизны.

## C1 (Метод Get)

Возвращает параметр координаты перегиба первой функции принадлежности.

```
double C1()
```

#### Возвращаемое значение

Значение координаты перегиба.

## C1 (Метод Set)

Устанавливает параметр координаты перегиба первой функции принадлежности.

```
void C1(
    const double c1          // значение координаты перегиба
)
```

#### Параметры

*c1*

[in] Значение координаты перегиба.

## C2 (Метод Get)

Возвращает параметр координаты перегиба второй функции принадлежности.

```
double C2()
```

#### Возвращаемое значение

Значение координаты перегиба.

## C2 (Метод Set)

Устанавливает параметр координаты перегиба второй функции принадлежности.

```
void C2(
    const double c2          // значение координаты перегиба
)
```

## Параметры

c2

[in] Значение координаты перегиба.

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(  
    const double x          // аргумент функции принадлежности  
)
```

## Параметры

x

[in] Аргумент функции принадлежности.

## Возвращаемое значение

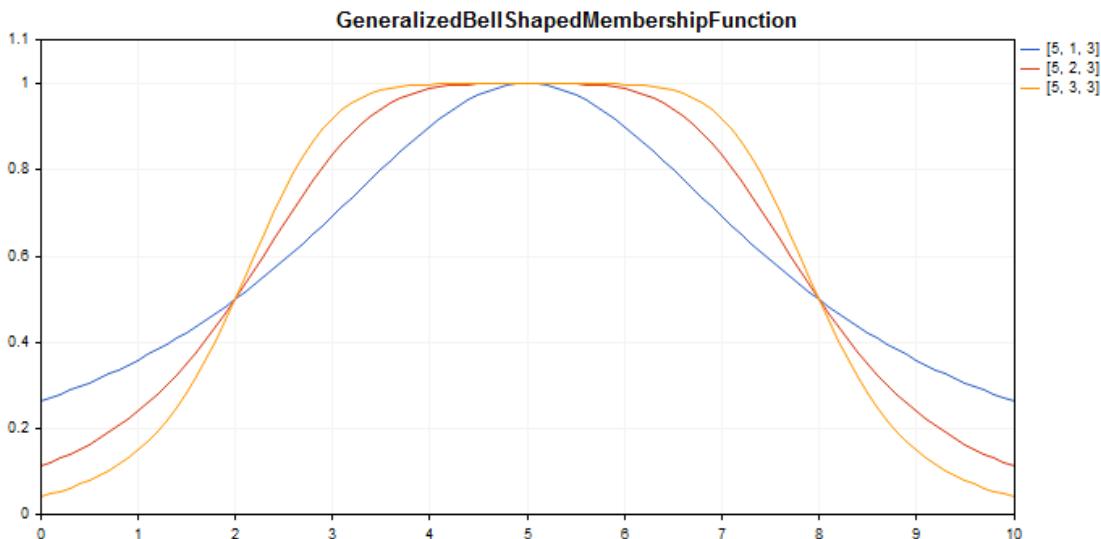
Значение функции принадлежности.

## CGeneralizedBellShapedMembershipFunction

Класс для реализации обобщенной колоколообразной функции принадлежности с параметрами A, B и C.

### Описание

Обобщенная колоколообразная функция принадлежности по форме похожа на гауссовские. На всей области определения функция является гладкой и принимает ненулевые значения.



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CGeneralizedBellShapedMembershipFuncion : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject  
IMembershipFunction  
CGeneralizedBellShapedMembershipFunction
```

### Методы класса

Метод класса	Описание
A	Возвращает и устанавливает коэффициент концентрации функции принадлежности.
B	Возвращает и устанавливает коэффициент крутизны функции принадлежности.

<u>C</u>	Возвращает и устанавливает координату максимума функции принадлежности.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Пример

```

//-----+
//|                               GeneralizedBellShapedMembershipFunction.mq5 |
//|                               Copyright 2016, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CGeneralizedBellShapedMembershipFunction func1(5, 1, 3);
CGeneralizedBellShapedMembershipFunction func2(5, 2, 3);
CGeneralizedBellShapedMembershipFunction func3(5, 3, 3);
//--- Create wrappers for membership functions
double GeneralizedBellShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double GeneralizedBellShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double GeneralizedBellShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0, "GeneralizedBellShapedMembershipFunction", 0, 30, 30, 780, 380))
{
    graphic.Attach(0, "GeneralizedBellShapedMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("GeneralizedBellShapedMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(GeneralizedBellShapedMembershipFunction1, 0.0, 10.0, 0.1, CURVE_LINES,
graphic.CurveAdd(GeneralizedBellShapedMembershipFunction2, 0.0, 10.0, 0.1, CURVE_LINES,
graphic.CurveAdd(GeneralizedBellShapedMembershipFunction3, 0.0, 10.0, 0.1, CURVE_LINES,
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);

```

```
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

## A (Метод Get)

Возвращает коэффициент концентрации функции принадлежности.

```
double A()
```

### Возвращаемое значение

Значение коэффициента концентрации.

## A (Метод Set)

Устанавливает коэффициент концентрации функции принадлежности.

```
void A(
    const double a // значение коэффициента концентрации
)
```

### Параметры

*a*

[in] Коэффициент концентрации функции принадлежности.

## B (Метод Get)

Возвращает коэффициент крутизны функции принадлежности.

```
double B()
```

### Возвращаемое значение

Значение коэффициента крутизны.

## B (Метод Set)

Устанавливает коэффициент крутизны функции принадлежности.

```
void B(
    const double b // значение коэффициента крутизны
)
```

)

**Параметры***b*

[in] Коэффициент крутизны функции принадлежности.

**C (Метод Get)**

Возвращает координату максимума функции принадлежности.

`double C()`**Возвращаемое значение**

Координата максимума функции принадлежности.

**C (Метод Set)**

Устанавливает координату максимума функции принадлежности.

`void C(  
 const double c // значение координаты максимума  
)`**Параметры***c*

[in] Координата максимума функции принадлежности.

**GetValue**

Рассчитывает значение функции принадлежности по указанному аргументу.

`double GetValue(  
 const x // аргумент функции принадлежности  
)`**Параметры***x*

[in] Аргумент функции принадлежности.

**Возвращаемое значение**

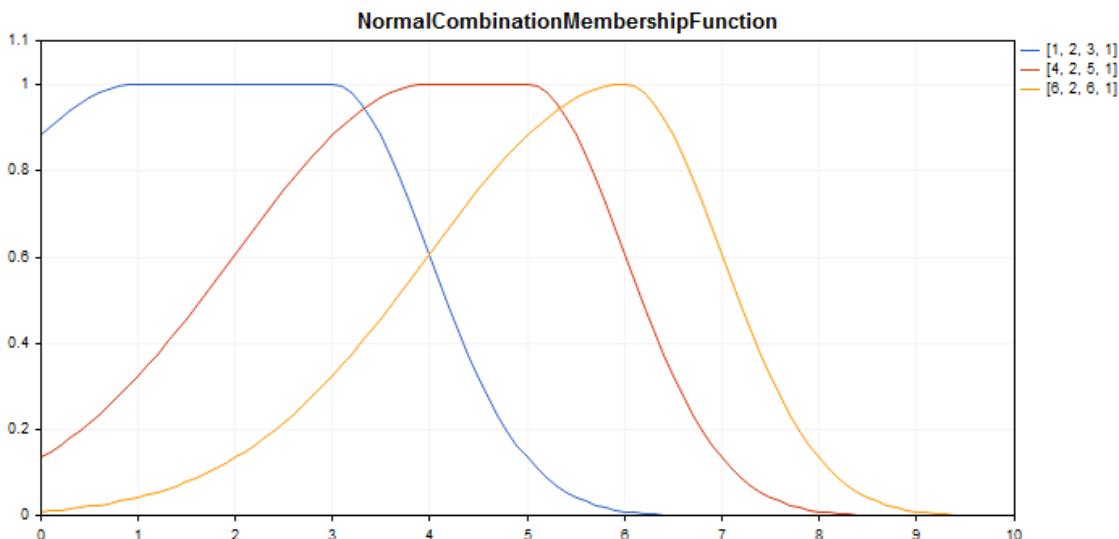
Значение функции принадлежности.

## CNormalCombinationMembershipFunction

Класс для реализации двухсторонней гауссовой функции принадлежности с параметрами B1, B2, Sigma1 и Sigma2.

### Описание

Двухсторонняя гауссовская функция принадлежности формируется с использованием распределения Гаусса. Она позволяет задавать асимметричные функции принадлежности. На всей области определения функция является гладкой и принимает ненулевые значения.



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CNormalCombinationMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject  
IMembershipFunction  
CNormalCombinationMembershipFunction
```

### Методы класса

Метод класса	Описание
<u>B1</u>	Возвращает и устанавливает значение первого центра функции принадлежности.

<u>B2</u>	Возвращает и устанавливает значение второго центра функции принадлежности.
<u>Sigma1</u>	Возвращает и устанавливает первый параметр кривизны функции принадлежности.
<u>Sigma2</u>	Возвращает и устанавливает второй параметр кривизны функции принадлежности.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

**Пример**

```

//+-----+
//|                               NormalCombinationMembershipFunction.mqh  |
//| Copyright 2016, MetaQuotes Software Corp. | |
//|                                         https://www.mql5.com | |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CNormalCombinationMembershipFunction func1(1,2,3,1);
CNormalCombinationMembershipFunction func2(4,2,5,1);
CNormalCombinationMembershipFunction func3(6,2,6,1);
//--- Create wrappers for membership functions
double NormalCombinationMembershipFunction1(double x) { return(func1.GetValue(x)); }
double NormalCombinationMembershipFunction2(double x) { return(func2.GetValue(x)); }
double NormalCombinationMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function                                |
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"NormalCombinationMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"NormalCombinationMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("NormalCombinationMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(NormalCombinationMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[1",
graphic.CurveAdd(NormalCombinationMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[4",
graphic.CurveAdd(NormalCombinationMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[6,

```

```
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

## B1 (Метод Get)

Возвращает значение первого центра функции принадлежности.

```
double B1()
```

### Возвращаемое значение

Значение первого центра функции принадлежности.

## B1 (Метод Set)

Устанавливает значение первого центра функции принадлежности.

```
void B1(
    const double b1           // значение первого центра
)
```

### Параметры

*b*

[in] Значение первого центра функции принадлежности.

## B2 (Метод Get)

Возвращает значение второго центра функции принадлежности.

```
double B2()
```

### Возвращаемое значение

Значение второго центра функции принадлежности.

## B2 (Метод Set)

Устанавливает значение второго центра функции принадлежности.

```
void B2(
    const double b2      // значение второго центра
)
```

#### Параметры

*b2*

[in] Значение второго центра функции принадлежности.

## Sigma1 (Метод Get)

Возвращает первый параметр кривизны функции принадлежности.

```
double Sigma1()
```

#### Возвращаемое значение

Значение первого параметра кривизны функции принадлежности.

## Sigma1 (Метод Set)

Устанавливает значение первого параметра кривизны функции принадлежности.

```
void Sigma1(
    const double sigma1      // значение первого параметра кривизны
)
```

#### Параметры

*sigma1*

[in] Первый параметр кривизны функции принадлежности.

## Sigma2 (Метод Get)

Возвращает второй параметр кривизны функции принадлежности.

```
double Sigma2()
```

#### Возвращаемое значение

Значение второго параметра кривизны функции принадлежности.

## Sigma2 (Метод Set)

Устанавливает значение второго параметра кривизны функции принадлежности.

```
void Sigma2(
```

```
const double sigma2           // значение второго параметра кривизны
)
```

### Параметры

*sigma2*

[in] Второй параметр кривизны функции принадлежности.

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(
    const x           // аргумент функции принадлежности
)
```

### Параметры

*x*

[in] Аргумент функции принадлежности.

### Возвращаемое значение

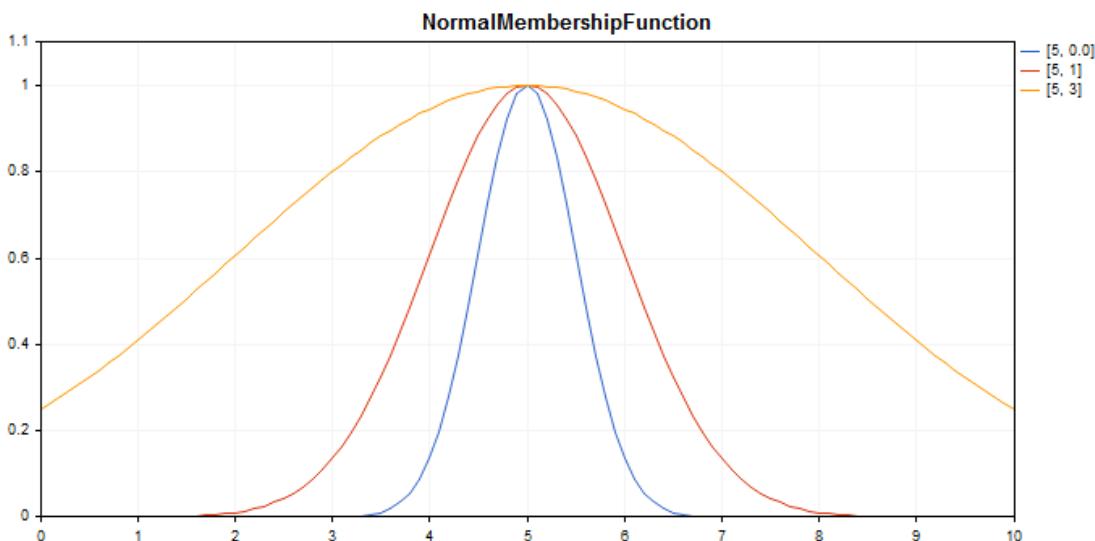
Значение функции принадлежности.

## CNormalMembershipFunction

Класс для реализации симметричной гауссовой функции принадлежности с параметрами В и Sigma.

### Описание

Симметричная гауссовская функция принадлежности формируется с использованием распределения Гаусса. На всей области определения функция является гладкой и принимает ненулевые значения.



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CNormalMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject  
IMembershipFunction  
CNormalMembershipFunction
```

### Методы класса

Метод класса	Описание
<u>B</u>	Возвращает и устанавливает центр функции принадлежности.

<u>Sigma</u>	Возвращает и устанавливает параметр кривизны функции принадлежности.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

**Методы унаследованные от CObject**Prev, Prev, Next, [Save](#), [Load](#), [Type](#), [Compare](#)**Пример**

```

//+-----+
//|                               NormalMembershipFunction.mq5 |
//|                               Copyright 2016, MetaQuotes Software Corp. |
//|                               https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CNormalMembershipFunction func1(5,0.5);
CNormalMembershipFunction func2(5,1);
CNormalMembershipFunction func3(5,3);
//--- Create wrappers for membership functions
double NormalMembershipFunction1(double x) { return(func1.GetValue(x)); }
double NormalMembershipFunction2(double x) { return(func2.GetValue(x)); }
double NormalMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"NormalMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"NormalMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("NormalMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(NormalMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[5, 0.0]");
graphic.CurveAdd(NormalMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[5, 1]");
graphic.CurveAdd(NormalMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[5, 3]");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
}

```

```
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

## B (Метод Get)

Возвращает центр функции принадлежности.

```
double B()
```

### Возвращаемое значение

Значение центра функции принадлежности.

## B (Метод Set)

Устанавливает значение центра функции принадлежности.

```
void B(
    const double b          // значение центра функции
)
```

### Параметры

*b*

[in] Значение центра функции принадлежности.

## Sigma (Метод Get)

Возвращает параметр кривизны функции принадлежности

```
double Sigma()
```

### Возвращаемое значение

Значение параметра кривизны функции принадлежности

## Sigma (Метод Set)

Устанавливает значение параметра кривизны функции принадлежности.

```
void Sigma(
    const double sigma        // значение параметра кривизны
)
```

### Параметры

*sigma*

[in] Параметр кривизны функции принадлежности.

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(  
    const double x          // аргумент  
)
```

### Параметры

x

[in] Аргумент функции принадлежности.

### Возвращаемое значение

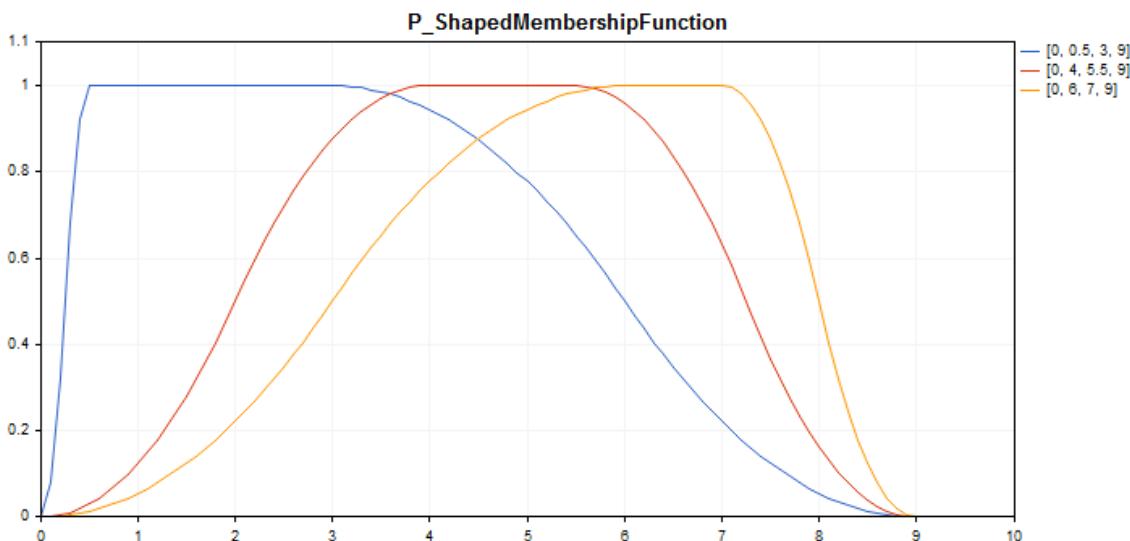
Значение функции принадлежности.

## CP\_ShapedMembershipFunction

Класс для реализации пи-подобной функции принадлежности с параметрами A, B, C и D.

### Описание

Пи-подобная функция принадлежности имеет вид криволинейной трапеции. Функция применяется для задания асимметричных функций принадлежности с плавным переходом от пессимистической к оптимистической оценке нечеткого числа.



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CP_ShapedMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject  
IMembershipFunction  
CP_ShapedMembershipFunction
```

### Методы класса

Метод класса	Описание
<u>A</u>	Возвращает и устанавливает параметр начала нечеткого множества.
<u>B</u>	Возвращает и устанавливает первый параметр ядра нечеткого множества.

<u>C</u>	Возвращает и устанавливает второй параметр ядра нечеткого множества.
<u>D</u>	Возвращает и устанавливает параметр конца нечеткого множества.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

**Пример**

```
//+-----+
//| P_ShapedMembershipFunction.mq5 |
//| Copyright 2016, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CP_ShapedMembershipFunction func1(0,0.5,3,9);
CP_ShapedMembershipFunction func2(0,4,5.5,9);
CP_ShapedMembershipFunction func3(0,6,7,9);
//--- Create wrappers for membership functions
double P_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double P_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double P_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function |
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"P_ShapedMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"P_ShapedMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("P_ShapedMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(P_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 0.5, 3,
graphic.CurveAdd(P_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 4, 5.5,
graphic.CurveAdd(P_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[0, 6, 7, 9]
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
```

```

graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}

```

## A (Метод Get)

Возвращает параметр начала нечеткого множества.

```
double A()
```

### Возвращаемое значение

Параметр начала нечеткого множества.

## A (Метод Set)

Устанавливает параметр начала нечеткого множества.

```

void A(
    const double a          // параметр начала нечеткого множества
)

```

### Параметры

*a*

[in] Параметр начала нечеткого множества.

## B (Метод Get)

Возвращает первый параметр ядра нечеткого множества.

```
double B()
```

### Возвращаемое значение

Первый параметр ядра нечеткого множества.

## B (Метод Set)

Устанавливает первый параметр ядра нечеткого множества.

```

void B(
    const double b          // значение первого параметра ядра нечеткого множества
)

```

## Параметры

*b*

[in] Первый параметр ядра нечеткого множества.

## C (Метод Get)

Возвращает второй параметр ядра нечеткого множества.

```
double C()
```

### Возвращаемое значение

Второй параметр ядра нечеткого множества.

## C (Метод Set)

Устанавливает второй параметр ядра нечеткого множества.

```
void C(
    const double c           // значение второго параметра ядра нечеткого множества
)
```

## Параметры

*c*

[in] Второй параметр ядра нечеткого множества.

## D (Метод Get)

Возвращает параметр конца нечеткого множества.

```
double D()
```

### Возвращаемое значение

Значение параметра конца нечеткого множества.

## D (Метод Set)

Устанавливает параметр конца нечеткого множества.

```
void D(
    const double d           // значение параметра конца нечеткого множества
)
```

## Параметры

*d*

[in] Значение параметра конца нечеткого множества.

## GetValue

Расчитывает значение функции принадлежности по указанному аргументу

```
double GetValue(  
    const double x  
)
```

#### Параметры

x

[in] аргумент функции принадлежности

#### Возвращаемое значение

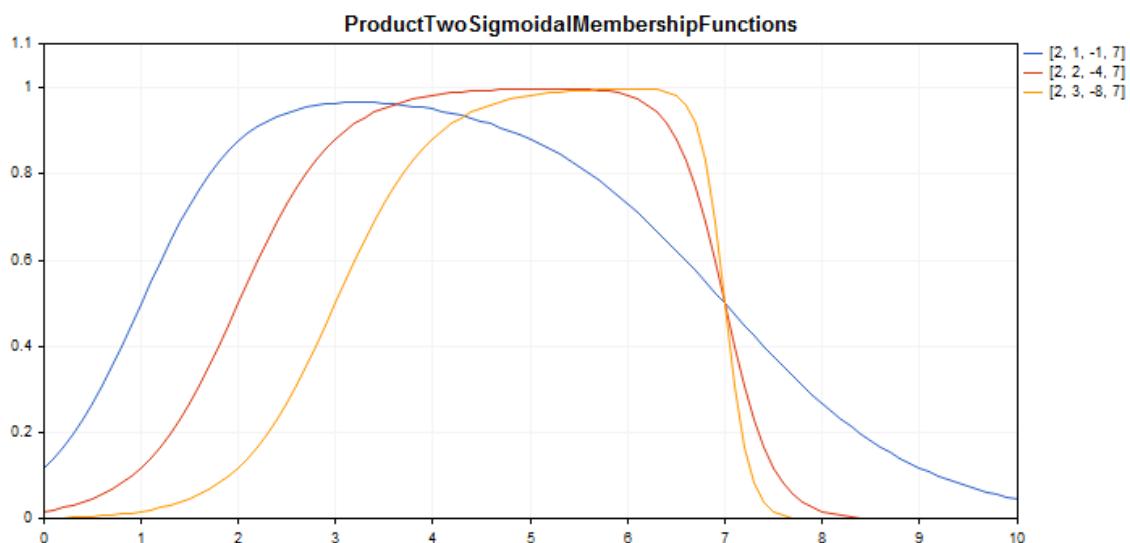
значение функции принадлежности

## CProductTwoSigmoidalMembershipFunction

Класс для реализации функции принадлежности в виде произведения двух сигмоидных функций с параметрами A1, A2, C1, C2.

### Описание

Произведение двух сигмоидных функций принадлежности применяется для задания гладких асимметричных функций. С его помощью можно создавать функции принадлежности со значениями, равными 1, начиная с некоторого значения аргумента. Подобные функции подходят, если необходимо задать лингвистические термы типа "короткий" или "длинный".



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CProductTwoSigmoidalMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

[CObject](#)

[IMembershipFunction](#)

CProductTwoSigmoidalMembershipFunctions

### Методы класса

Метод класса	Описание
<a href="#">A1</a>	Возвращает и устанавливает коэффициент крутизны первой функции принадлежности.

<u>A2</u>	Возвращает и устанавливает коэффициент крутизны второй функции принадлежности.
<u>C1</u>	Возвращает параметр координаты перегиба первой функции принадлежности.
<u>C2</u>	Возвращает параметр координаты перегиба второй функции принадлежности.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Пример

```

//+-----+
//|                               ProductTwoSigmoidalMembershipFunctions.mq5 |
//|                               Copyright 2016, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CProductTwoSigmoidalMembershipFunctions func1(2,1,-1,7);
CProductTwoSigmoidalMembershipFunctions func2(2,2,-4,7);
CProductTwoSigmoidalMembershipFunctions func3(2,3,-8,7);
//--- Create wrappers for membership functions
double ProductTwoSigmoidalMembershipFunctions1(double x) { return(func1.GetValue(x)); }
double ProductTwoSigmoidalMembershipFunctions2(double x) { return(func2.GetValue(x)); }
double ProductTwoSigmoidalMembershipFunctions3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"ProductTwoSigmoidalMembershipFunctions",0,30,30,780,380))
{
    graphic.Attach(0,"ProductTwoSigmoidalMembershipFunctions");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("ProductTwoSigmoidalMembershipFunctions");
graphic.BackgroundMainSize(16);
//--- create curve

```

```

graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions1,0.0,10.0,0.1,CURVE_LINES,
graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions2,0.0,10.0,0.1,CURVE_LINES,
graphic.CurveAdd(ProductTwoSigmoidalMembershipFunctions3,0.0,10.0,0.1,CURVE_LINES,
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}

```

## A1 (Метод Get)

Возвращает коэффициент крутизны первой функции принадлежности.

```
double A1()
```

### Возвращаемое значение

Коэффициент крутизны первой функции принадлежности.

## A1 (Метод Set)

Устанавливает коэффициент крутизны первой функции принадлежности.

```

void A1(
    const double a1           // коэффициент крутизны первой функции принадлежности
)

```

### Параметры

*a1*

[in] Коэффициент крутизны первой функции принадлежности.

## A2 (Метод Get)

Возвращает коэффициент крутизны второй функции принадлежности.

```
double A2()
```

### Возвращаемое значение

Коэффициент крутизны второй функции принадлежности.

## A2 (Метод Set)

Устанавливает коэффициент крутизны второй функции принадлежности.

```
void A2(
    const double a2          // коэффициент крутизны второй функции принадлежности
)
```

#### Параметры

*a2*

[in] Коэффициент крутизны второй функции принадлежности.

## C1 (Метод Get)

Возвращает параметр координаты перегиба первой функции принадлежности.

```
double C1()
```

#### Возвращаемое значение

Координата перегиба первой функции принадлежности.

## C1 (Метод Set)

Устанавливает координату перегиба первой функции принадлежности.

```
void C1(
    const double c1          // координата перегиба первой функции принадлежности
)
```

#### Параметры

*c1*

[in] Координата перегиба первой функции принадлежности.

## C2 (Метод Get)

Возвращает параметр координаты перегиба второй функции принадлежности.

```
double C2()
```

#### Возвращаемое значение

Координата перегиба второй функции принадлежности.

## C2 (Метод Set)

Устанавливает координату перегиба второй функции принадлежности.

```
void C2(
    const double c2          // координата перегиба второй функции принадлежности
)
```

#### Параметры

*c2*

[in] Координата перегиба второй функции принадлежности.

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(  
    const x          // аргумент функции принадлежности  
)
```

### Параметры

x

[in] Аргумент функции принадлежности.

### Возвращаемое значение

Значение функции принадлежности.

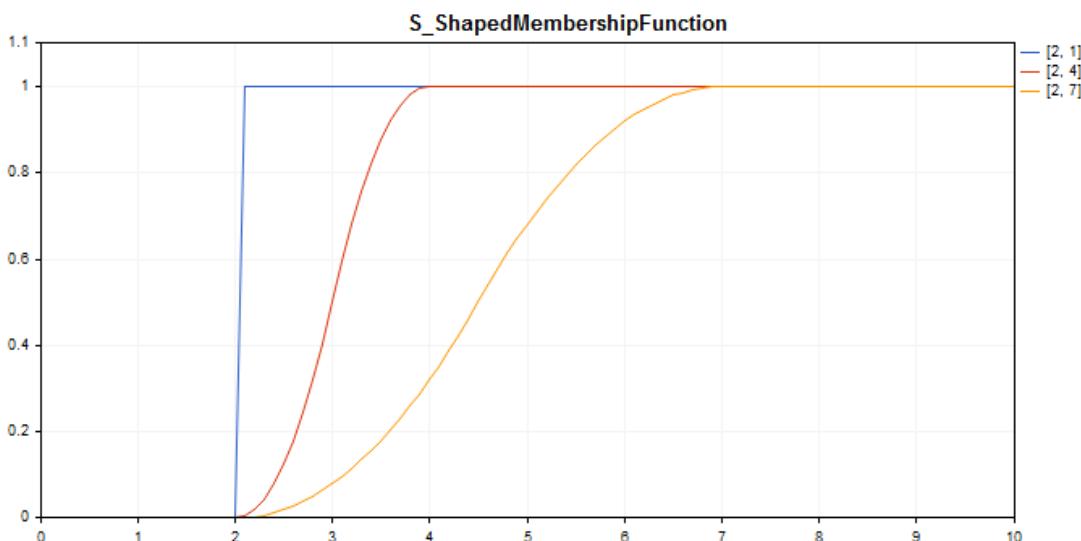
## CS\_ShapedMembershipFunction

Класс для реализации S-подобной функции принадлежности с параметрами А и В.

### Описание

Функция задает S-подобную двухпараметрическую функцию принадлежности. Это неубывающая функция, принимающая значения от 0 до 1. Параметры А и В определяют интервал, внутри которого функция возрастает по нелинейной траектории от 0 до 1.

С помощью этой функции представлены нечеткие множества типа "очень высокий" (то есть, задаются неубывающие функции принадлежности с насыщением).



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CS_ShapedMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject  
IMembershipFunction  
CS_ShapedMembershipFunction
```

### Методы класса

Метод класса	Описание
<a href="#">A</a>	Возвращает и устанавливает параметр начала интервала возрастания.

<u>B</u>	Возвращает и устанавливает первый параметр ядра нечеткого множества.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

**Методы унаследованные от CObject**Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)**Пример**

```

//-----+
//|                               S_ShapedMembershipFunction.mq5 |
//|           Copyright 2016, MetaQuotes Software Corp. |
//|                           https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CS_ShapedMembershipFunction func1(2,1);
CS_ShapedMembershipFunction func2(2,4);
CS_ShapedMembershipFunction func3(2,7);
//--- Create wrappers for membership functions
double S_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double S_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double S_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0, "S_ShapedMembershipFunction", 0, 30, 30, 780, 380))
{
    graphic.Attach(0, "S_ShapedMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("S_ShapedMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(S_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[2, 1]");
graphic.CurveAdd(S_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[2, 4]");
graphic.CurveAdd(S_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 7]");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
}

```

```
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

## A (Метод Get)

Возвращает параметр начала интервала возрастания.

```
double A()
```

### Возвращаемое значение

Параметр начала интервала возрастания.

## A (Метод Set)

Устанавливает параметр начала интервала возрастания.

```
void A(
    const double a // параметр начала интервала возрастания
)
```

### Параметры

*a*

[in] Параметр начала интервала возрастания.

## B (Метод Get)

Возвращает первый параметр ядра нечеткого множества.

```
double B()
```

### Возвращаемое значение

Первый параметр ядра нечеткого множества.

## B (Метод Set)

Устанавливает первый параметр ядра нечеткого множества.

```
void B(
    const double b // первый параметр ядра нечеткого множества
)
```

### Параметры

*b*

[in] Первый параметр ядра нечеткого множества.

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(  
    const x          // аргумент функции принадлежности  
)
```

### Параметры

x

[in] Аргумент функции принадлежности.

### Возвращаемое значение

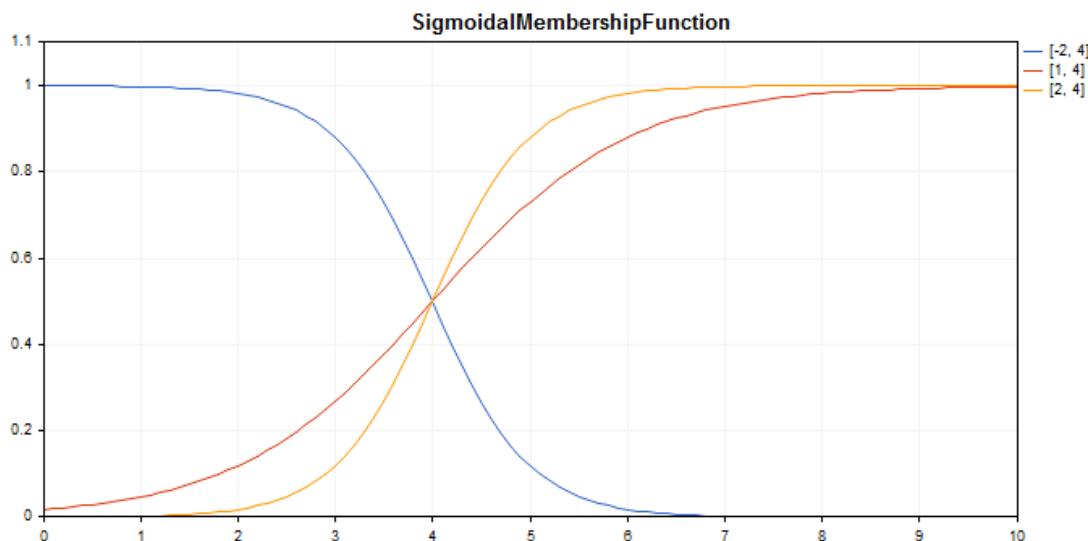
Значение функции принадлежности.

## CSigmoidalMembershipFunction

Класс для реализации сигмоидной функции принадлежности с параметрами А и С.

### Описание

Сигмоидная функция применяется для задания монотонных функций принадлежности. С ее помощью можно создавать функции принадлежности со значениями, равными 1, начиная с некоторого значения аргумента. Подобные функции подходят, если необходимо задать лингвистические термы типа "короткий" или "длинный".



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CSigmoidalMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject  
IMembershipFunction  
CSigmoidalMembershipFunction
```

### Методы класса

Метод класса	Описание
A	Возвращает и устанавливает коэффициент крутизны функции принадлежности.

<u>C</u>	Возвращает и устанавливает координаты перегиба функции принадлежности.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Пример

```
//+-----+
//|                               SigmoidalMembershipFunction.mq5 |
//| Copyright 2016, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CSigmoidalMembershipFunction func1(-2, 4);
CSigmoidalMembershipFunction func2(1, 4);
CSigmoidalMembershipFunction func3(2, 4);
//--- Create wrappers for membership functions
double SigmoidalMembershipFunction1(double x) { return(func1.GetValue(x)); }
double SigmoidalMembershipFunction2(double x) { return(func2.GetValue(x)); }
double SigmoidalMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0, "SigmoidalMembershipFunction", 0, 30, 30, 780, 380))
{
    graphic.Attach(0, "SigmoidalMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("SigmoidalMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(SigmoidalMembershipFunction1, 0.0, 10.0, 0.1, CURVE_LINES, "[ -2, 4 ]");
graphic.CurveAdd(SigmoidalMembershipFunction2, 0.0, 10.0, 0.1, CURVE_LINES, "[ 1, 4 ]");
graphic.CurveAdd(SigmoidalMembershipFunction3, 0.0, 10.0, 0.1, CURVE_LINES, "[ 2, 4 ]");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
```

```

graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}

```

## A (Метод Get)

Возвращает коэффициент крутизны функции принадлежности.

```
double A()
```

### Возвращаемое значение

Коэффициент крутизны функции принадлежности.

## A (Метод Set)

Устанавливает коэффициент крутизны функции принадлежности.

```

void A(
    const double a          // коэффициент крутизны функции принадлежности
)

```

### Параметры

*a*

[in] Коэффициент крутизны функции принадлежности.

## C (Метод Get)

Возвращает параметр координаты перегиба функции принадлежности.

```
double C()
```

### Возвращаемое значение

Координата перегиба функции принадлежности.

## C (Метод Set)

Устанавливает координату перегиба функции принадлежности.

```

void C(
    const double c          // координата перегиба функции принадлежности
)

```

### Параметры

*C*

[in] Координата перегиба функции принадлежности.

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(  
    const x // аргумент функции принадлежности  
)
```

### Параметры

*x*

[in] Аргумент функции принадлежности.

### Возвращаемое значение

Значение функции принадлежности.

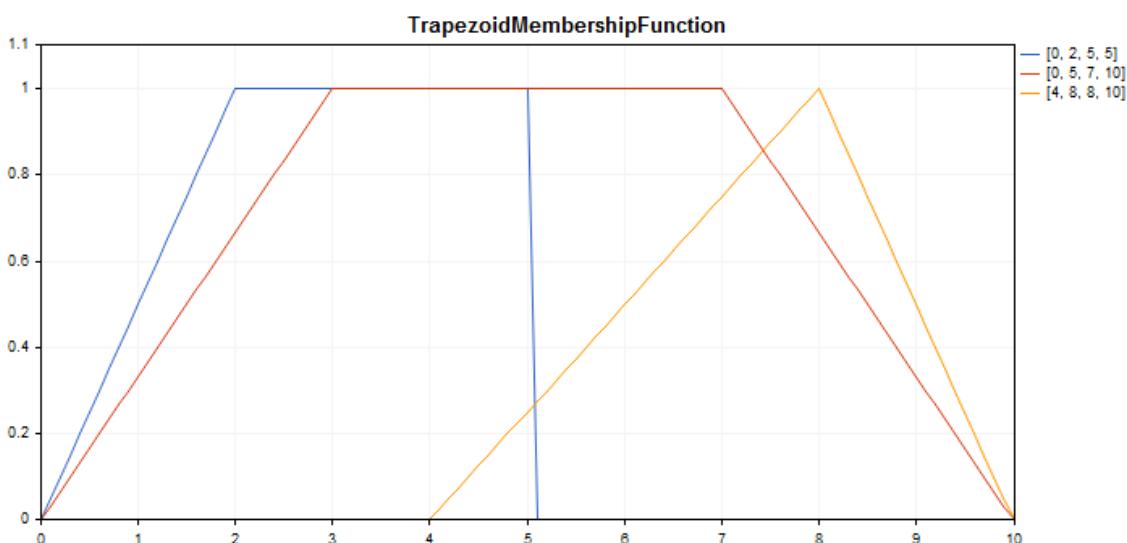
## CTrapezoidMembershipFunction

Класс для реализации трапециевидной функции принадлежности с параметрами X1, X2, X3 и X4.

### Описание

Функция формируется с использованием кусочно-линейной аппроксимации. Это обобщение треугольной функции, она позволяет задавать ядро нечеткого множества в виде интервала. Такая функция принадлежности делает возможной удобную интерпретацию оптимистической/пессимистической оценки.

Применяется для задания асимметричных функций принадлежности переменных, наиболее возможные значения которых принимаются на некотором интервале.



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CTrapizoidMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject  
IMembershipFunction  
CTrapizoidMembershipFunction
```

### Методы класса

Метод класса	Описание
--------------	----------

<u>X1</u>	Возвращает и устанавливает значение первой точки на абсциссе.
<u>X2</u>	Возвращает и устанавливает значение второй точки на абсциссе.
<u>X3</u>	Возвращает и устанавливает значение третьей точки на абсциссе.
<u>X4</u>	Возвращает и устанавливает значение четвертой точки на абсциссе.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Пример

```
//+-----+
//|                               TrapezoidMembershipFunction.mq5  |
//|                               Copyright 2016, MetaQuotes Software Corp.  |
//|                               https://www.mql5.com  |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CTrapezoidMembershipFunction func1(0,2,5,5);
CTrapezoidMembershipFunction func2(0,3,7,10);
CTrapezoidMembershipFunction func3(4,8,8,10);
//--- Create wrappers for membership functions
double TrapezoidMembershipFunction1(double x) { return(func1.GetValue(x)); }
double TrapezoidMembershipFunction2(double x) { return(func2.GetValue(x)); }
double TrapezoidMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"TrapezoidMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"TrapezoidMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("TrapezoidMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
```

```

graphic.CurveAdd(TrapezoidMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 2, 5, 5];
graphic.CurveAdd(TrapezoidMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 5, 7, 1];
graphic.CurveAdd(TrapezoidMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[4, 8, 8, 1];
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}

```

## X1 (Метод Get)

Возвращает значение первой точки на абсциссе.

```
double X1()
```

### Возвращаемое значение

Значение первой точки на абсциссе.

## X1 (Метод Set)

Устанавливает значение первой точки на абсциссе.

```

void X1(
    const double x1           // значение первой точки на абсциссе
)

```

### Параметры

x1

[in] Значение первой точки на абсциссе.

## X2 (Метод Get)

Возвращает значение второй точки на абсциссе.

```
double X2()
```

### Возвращаемое значение

Значение второй точки на абсциссе.

## X2 (Метод Set)

Устанавливает значение второй точки на абсциссе.

```
void X2(
    const double x2          // значение второй точки на абсциссе
)
```

#### Параметры

x2

[in] Значение второй точки на абсциссе.

## X3 (Метод Get)

Возвращает значение третьей точки на абсциссе.

```
double X3()
```

#### Возвращаемое значение

Значение третьей точки на абсциссе.

## X3 (Метод Set)

Устанавливает значение третьей точки на абсциссе.

```
void X3(
    const double x3          // значение третьей точки на абсциссе
)
```

#### Параметры

x3

[in] Значение третьей точки на абсциссе.

## X4 (Метод Get)

Возвращает значение четвертой точки на абсциссе.

```
double X4()
```

#### Возвращаемое значение

Значение четвертой точки на абсциссе.

## X4 (Метод Set)

Устанавливает значение четвертой точки на абсциссе.

```
void X4(
    const double x4          // значение четвертой точки на абсциссе
)
```

#### Параметры

x4

[in] Значение четвертой точки на абсциссе.

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(  
    const x          // аргумент функции принадлежности  
)
```

### Параметры

x

[in] Аргумент функции принадлежности.

### Возвращаемое значение

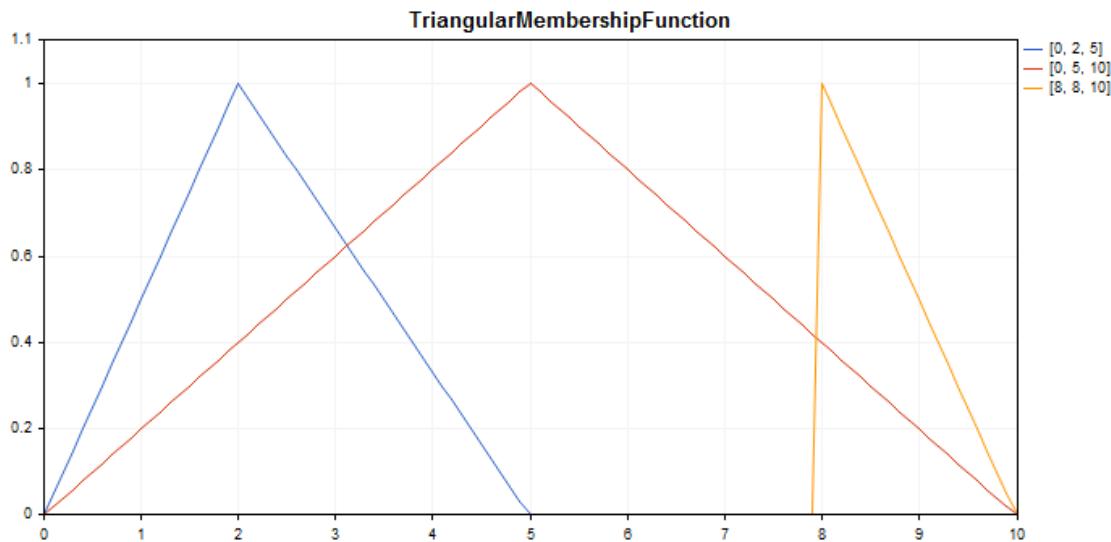
Значение функции принадлежности.

## CTriangularMembershipFunction

Класс для реализации треугольной функции принадлежности с параметрами X1, X2, X3.

### Описание

Функция задает функцию принадлежности в виде треугольника. Простая и наиболее часто применяемая функция принадлежности.



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CTriangularMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject  
IMembershipFunction  
CTriangularMembershipFunction
```

### Методы класса

Метод класса	Описание
<u>X1</u>	Возвращает значение первой точки на абсциссе.
<u>X2</u>	Возвращает значение второй точки на абсциссе.

<u>X3</u>	Возвращает значение третьей точки на абсциссе.
<u>ToNormalMF</u>	Преобразует треугольную функцию принадлежности в гауссову.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Пример

```
//+-----+
//|                                         TriangularMembershipFunction.mq5 |
//|                                         Copyright 2016, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2016, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CTriangularMembershipFunction func1(0,2,5);
CTriangularMembershipFunction func2(0,5,10);
CTriangularMembershipFunction func3(8,8,10);
//--- Create wrappers for membership functions
double TriangularMembershipFunction1(double x) { return(func1.GetValue(x)); }
double TriangularMembershipFunction2(double x) { return(func2.GetValue(x)); }
double TriangularMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0,"TriangularMembershipFunction",0,30,30,780,380))
{
    graphic.Attach(0,"TriangularMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("TriangularMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(TriangularMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[0, 2, 5]");
graphic.CurveAdd(TriangularMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[0, 5, 10]");
```

```

graphic.CurveAdd(TriangularMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[8, 8, 10]
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}

```

## X1 (Метод Get)

Возвращает значение первой точки на абсциссе.

```
double x1()
```

### Возвращаемое значение

Значение первой точки на абсциссе.

## X1 (Метод Set)

Устанавливает значение первой точки на абсциссе.

```

void x1(
    const double x1          // значение первой точки на абсциссе
)

```

### Параметры

x1

[in] Значение первой точки на абсциссе.

## X2 (Метод Get)

Возвращает значение второй точки на абсциссе.

```
double x2()
```

### Возвращаемое значение

Значение второй точки на абсциссе.

## X2 (Метод Set)

Устанавливает значение второй точки на абсциссе.

```
void x2(
    const double x2      // значение второй точки на абсциссе
)
```

**Параметры**

x2

[in] Значение второй точки на абсциссе.

**X3 (Метод Get)**

Возвращает значение третьей точки на абсциссе.

```
double x3()
```

**Возвращаемое значение**

Значение третьей точки на абсциссе.

**X3 (Метод Set)**

Устанавливает значение третьей точки на абсциссе.

```
void x3(
    const double x3      // значение третьей точки на абсциссе
)
```

**Параметры**

x3

[in] Значение третьей точки на абсциссе.

**ToNormalMF**

Преобразует треугольную функцию принадлежности в Гауссовскую.

```
CNormalMembershipFunction* ToNormalMF()
```

**Возвращаемое значение**Указатель на [Гауссовскую функцию принадлежности](#).**GetValue**

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(
    const x      // аргумент функции принадлежности
)
```

**Параметры**

x

[in] Аргумент функции принадлежности.

#### Возвращаемое значение

Значение функции принадлежности.

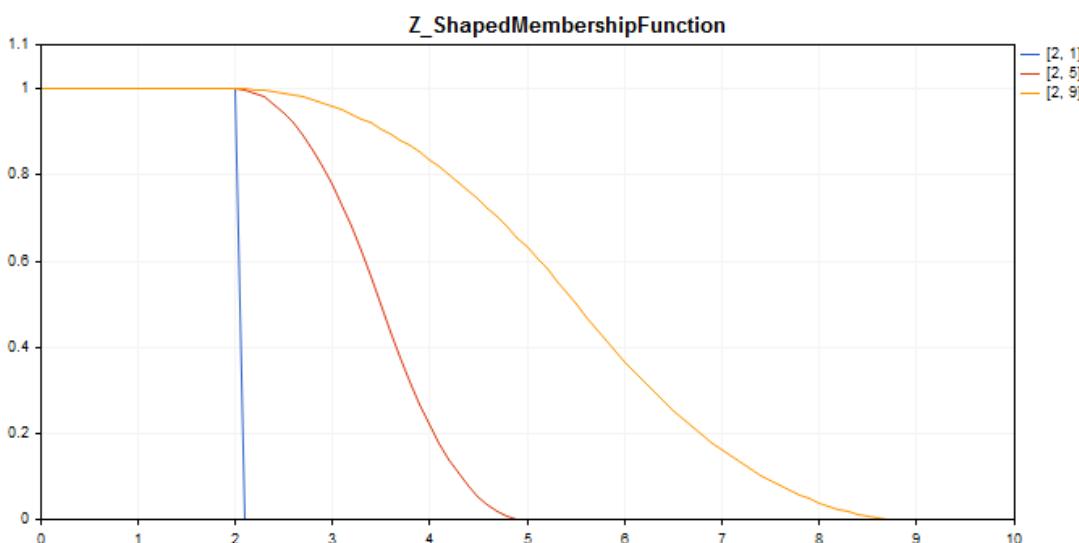
## CZ\_ShapedMembershipFunction

Класс для реализации z-подобной функции принадлежности с параметрами А и В.

### Описание

Функция задает z-подобную двухпараметрическую функцию принадлежности. Это невозрастающая функция принадлежности, принимающая значения от 1 до 0. Параметры функции определяют интервал, внутри которого функция нелинейно убывает от 0 до 1.

С помощью функции могут быть представлены нечеткие множества типа "очень низкий". То есть, она задает невозрастающие функции принадлежности с насыщением.



[Пример кода](#) для построения этого графика приведен ниже.

### Декларация

```
class CZ_ShapedMembershipFunction : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

```
CObject  
IMembershipFunction  
CZ_ShapedMembershipFunction
```

### Методы класса

Метод класса	Описание
<a href="#">A</a>	Возвращает и устанавливает параметр начала интервала убывания.

<u>B</u>	Возвращает и устанавливает параметр конца интервала убывания.
<u>GetValue</u>	Рассчитывает значение функции принадлежности по указанному аргументу.

**Методы унаследованные от CObject**Prev, Prev, Next, Next, [Save](#), [Load](#), [Type](#), [Compare](#)**Пример**

```

//-----+
//|                               Z_ShapedMembershipFunction.mq5 |
//|           Copyright 2016, MetaQuotes Software Corp. |
//|                           https://www.mql5.com |
//+-----+
#include <Math\Fuzzy\membershipfunction.mqh>
#include <Graphics\Graphic.mqh>
//--- Create membership functions
CZ_ShapedMembershipFunction func1(2,1);
CZ_ShapedMembershipFunction func2(2,5);
CZ_ShapedMembershipFunction func3(2,9);
//--- Create wrappers for membership functions
double Z_ShapedMembershipFunction1(double x) { return(func1.GetValue(x)); }
double Z_ShapedMembershipFunction2(double x) { return(func2.GetValue(x)); }
double Z_ShapedMembershipFunction3(double x) { return(func3.GetValue(x)); }
//+-----+
//| Script program start function
//+-----+
void OnStart()
{
//--- create graphic
CGraphic graphic;
if(!graphic.Create(0, "Z_ShapedMembershipFunction", 0, 30, 30, 780, 380))
{
    graphic.Attach(0, "Z_ShapedMembershipFunction");
}
graphic.HistoryNameWidth(70);
graphic.BackgroundMain("Z_ShapedMembershipFunction");
graphic.BackgroundMainSize(16);
//--- create curve
graphic.CurveAdd(Z_ShapedMembershipFunction1,0.0,10.0,0.1,CURVE_LINES,"[2, 1]");
graphic.CurveAdd(Z_ShapedMembershipFunction2,0.0,10.0,0.1,CURVE_LINES,"[2, 5]");
graphic.CurveAdd(Z_ShapedMembershipFunction3,0.0,10.0,0.1,CURVE_LINES,"[2, 9]");
//--- sets the X-axis properties
graphic.XAxis().AutoScale(false);
graphic.XAxis().Min(0.0);
graphic.XAxis().Max(10.0);
graphic.XAxis().DefaultStep(1.0);
}

```

```
//--- sets the Y-axis properties
graphic.YAxis().AutoScale(false);
graphic.YAxis().Min(0.0);
graphic.YAxis().Max(1.1);
graphic.YAxis().DefaultStep(0.2);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

## A (Метод Get)

Возвращает параметр начала интервала убывания.

```
double A()
```

### Возвращаемое значение

Параметр начала интервала убывания.

## A (Метод Set)

Устанавливает параметр начала интервала убывания.

```
void A(
    const double a // параметр начала интервала убывания
)
```

### Параметры

*a*

[in] Параметр начала интервала убывания.

## B (Метод Get)

Возвращает параметр конца интервала убывания.

```
double B()
```

### Возвращаемое значение

Параметр конца интервала убывания.

## B (Метод Set)

Устанавливает параметр конца интервала убывания.

```
void B(
    const double b // параметр конца интервала убывания
)
```

### Параметры

*b*

[in] Параметр конца интервала убывания.

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(  
    const x // аргумент функции принадлежности  
)
```

### Параметры

x

[in] Аргумент функции принадлежности.

### Возвращаемое значение

Значение функции принадлежности.

## IMembershipFunction

Базовый класс для всех классов функций принадлежности.

### Декларация

```
class CZ_ShapedMembershipFuncion : public IMembershipFunction
```

### Заголовок

```
#include <Math\Fuzzy\membershipfunction.mqh>
```

### Иерархия наследования

[CObject](#)

IMembershipFunction

### Прямые потомки

<a href="#">CCompositeMembershipFunction</a> ,	<a href="#">CConstantMembershipFunction</a> ,
<a href="#">CDifferencTwoSigmoidalMembershipFunction</a> ,	<a href="#">CGeneralizedBellShapedMembershipFunction</a> ,
<a href="#">CNormalCombinationMembershipFunction</a> ,	<a href="#">CNormalMembershipFunction</a> ,
<a href="#">CP_ShapedMembershipFunction</a> ,	<a href="#">CProductTwoSigmoidalMembershipFunctions</a> ,
<a href="#">CS_ShapedMembershipFunction</a> , <a href="#">CSigmoidalMembershipFunction</a> , <a href="#">CTrapezoidMembershipFunction</a> ,	<a href="#">CTriangularMembershipFunction</a> , <a href="#">CZ_ShapedMembershipFunction</a>

### Методы класса

Метод класса	Описание
<a href="#">GetValue</a>	Рассчитывает значение функции принадлежности по указанному аргументу.

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## GetValue

Рассчитывает значение функции принадлежности по указанному аргументу.

```
double GetValue(
    const x // аргумент функции принадлежности
)
```

### Параметры

x

[in] Аргумент функции принадлежности.

### Возвращаемое значение

Значение функции принадлежности.

## Правила для нечетких систем

**Нечеткая система** (система нечеткого логического вывода) — это получение заключения в виде нечеткого множества, соответствующего текущим значениям входов, с использованием совокупности нечетких правил и нечетких операций.

**Нечеткие правила** определяют взаимосвязь между входами и выходами исследуемого объекта. Количество правил в системе не ограничено. Обобщенный формат нечетких правил таков:

*если условие (посылка) правила, то заключение правила.*

*Условие правила* характеризует текущее состояние объекта. *Заключение правила* характеризует то, как условие повлияет на объект.

Класс правил для нечетких систем	Описание
<a href="#"><u>CMamdaniFuzzyRule</u></a>	Класс для реализации нечеткого логического правила для алгоритма Мамдани
<a href="#"><u>CSugenoFuzzyRule</u></a>	Класс для реализации нечеткого логического правила для алгоритма Сугено
<a href="#"><u>CSingleCondition</u></a>	Класс задает нечеткое условие, выраженное парой "Нечеткая переменная – Нечеткий терм".
<a href="#"><u>CConditions</u></a>	Класс задает набор нечетких условий, связанных оператором.
<a href="#"><u>CGenericFuzzyRule</u></a>	Базовый класс для реализации обоих типов нечетких правил.

## CMamdaniFuzzyRule

Нечеткий логический вывод Мамдани – один из двух основных типов нечетких систем. Значения выходной переменной в нем задаются нечеткими термами.

### Описание

Нечеткое логическое правило для алгоритма Мамдани можно описать следующим выражением:

$if(X_1 \text{ is } a_1) \wedge (X_2 \text{ is } a_2) \wedge \dots \wedge (X_n \text{ is } a_n) \text{ then } (Y \text{ is } d)(W)$

где:

- X = (X<sub>1</sub>, X<sub>2</sub>, X<sub>3</sub> ... X<sub>n</sub>) – вектор входных переменных;
- Y – выходная переменная;
- a = (a<sub>1</sub>, a<sub>2</sub>, a<sub>3</sub> ... a<sub>n</sub>) – вектор значений входных переменных;
- d – значение выходной переменной;
- W – вес правила.

### Декларация

```
class CMamdaniFuzzyRule : public CGenericFuzzyRule
```

### Заголовок

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

### Иерархия наследования

```
CObject
  IParsableRule
    CGenericFuzzyRule
      CMamdaniFuzzyRule
```

### Методы класса

Метод класса	Описание
<a href="#">Conclusion</a>	Возвращает и устанавливает заключение нечёткого правила Мамдани.
<a href="#">Weight</a>	Возвращает и устанавливает вес нечёткого правила Мамдани.

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CGenericFuzzyRule

[Condition](#), [Condition](#), [CreateCondition](#), [CreateCondition](#), [CreateCondition](#)

## Conclusion (Метод Get)

Возвращает заключение нечёткого правила Мамдани.

```
CSingleConditon* Conclusion()
```

### Возвращаемое значение

Заключение нечеткого правила Мамдани.

## Conclusion (Метод Set)

Устанавливает заключение нечёткого правила Мамдани.

```
void Conclusion(
    CSingleConditon* value      // заключение нечёткого правила Мамдани
)
```

### Параметры

*value*

[in] Заключение нечёткого правила Мамдани.

## Weight (Метод Get)

Возвращает вес нечёткого правила Мамдани.

```
double Weight()
```

### Возвращаемое значение

Вес нечёткого правила Мамдани.

## Weight (Метод Set)

Устанавливает вес нечёткого правила Мамдани.

```
void Weight(
    const double value      // вес нечёткого правила Мамдани
)
```

### Параметры

*value*

[in] Вес нечёткого правила Мамдани.

## CSugenoFuzzyRule

Нечеткий логический вывод Сугено – один из двух основных типов нечетких систем. Значения выходной переменной в нем задаются как линейная комбинация входных переменных.

### Описание

Отличие от правила Мамдани заключается в том, что значение выходной переменной задается не нечетким термом, а линейной функцией от входных переменных. Нечеткое логическое правило для алгоритма Сугено можно описать следующим выражением:

$$if(X_1 \text{ is } a_1) \wedge (X_2 \text{ is } a_2) \wedge \dots \wedge (X_n \text{ is } a_n) \text{ then } (Y = b_0 + b_1 \cdot X_1 + b_2 \cdot X_2 + \dots + b_n \cdot X_n)(W)$$

где:

- X = (X1, X2, X3 ... Xn) – вектор входных переменных;
- Y – выходная переменная;
- a = (a1, a2, a3 ... an) – вектор значений входных переменных;
- b = (b1, b2, b3 ... bn) – коэффициент при свободном члене в линейной функции для выходного значения
- W – вес правила.

### Декларация

```
class CSugenoFuzzyRule : public CGenericFuzzyRule
```

### Заголовок

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

### Иерархия наследования

```
CObject
  IParsableRule
    CGenericFuzzyRule
      CSugenoFuzzyRule
```

### Методы класса

Метод класса	Описание
<a href="#">Conclusion</a>	Возвращает и устанавливает заключение нечёткого правила Сугено.

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CGenericFuzzyRule

[Condition](#), [Condition](#), [CreateCondition](#), [CreateCondition](#), [CreateCondition](#)

## Conclusion (Метод Get)

Возвращает заключение нечёткого правила Сугено.

```
CSingleCondition* Conclusion()
```

### Возвращаемое значение

Заключение нечёткого правила Сугено.

## Conclusion (Метод Set)

Устанавливает заключение нечёткого правила Сугено.

```
void Conclusion(  
    CSingleCondition* value          // заключение нечёткого правила Сугено  
)
```

### Параметры

*value*

[in] Заключение нечёткого правила Сугено.

## CSingleCondition

Класс задает нечеткое условие, выраженное парой "Нечеткая переменная – Нечеткий терм".

### Описание

По нечеткому условию одной переменной соответствует один терм. Нечеткое условие можно описать следующим выражением:  $X \text{ is } a$ ,

где:

- $X$  – нечеткая переменная;
- $a$  – значение нечеткой переменной (нечеткий терм).

### Декларация

```
class CSingleCondition : public ICondition
```

### Заголовок

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

### Иерархия наследования

[CObject](#)

ICondition

CSingleCondition

### Прямые потомки

CFuzzyCondition

### Методы класса

Метод класса	Описание
<a href="#"><u>Not</u></a>	Возвращает и устанавливает флаг, указывающий на то, применить ли к этому условию отрицание.
<a href="#"><u>Term</u></a>	Возвращает и устанавливает нечёткий терм для данного условия.
<a href="#"><u>Var</u></a>	Возвращает и устанавливает нечёткую переменную для данного условия.

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Not (Метод Get)

Возвращает флаг, указывающий на то, применить ли к этому условию отрицание.

```
bool Not()
```

#### Возвращаемое значение

Значение флага.

### Not (Метод Set)

Устанавливает флаг, указывающий на то, применить ли к этому условию отрицание.

```
void Not(
    bool not      // значение флага
)
```

#### Параметры

*not*

[in] Значение флага.

### Term (Метод Get)

Возвращает нечёткий терм для данного условия.

```
INamedValue* Term()
```

#### Возвращаемое значение

Нечёткий терм для данного условия.

### Term (Метод Set)

Устанавливает нечёткий терм для данного условия.

```
void Term(
    INamedValue*& value      // нечёткий терм для данного условия
)
```

#### Параметры

*value*

[in] Нечёткий терм для данного условия.

### Var (Метод Get)

Возвращает нечёткую переменную для данного условия.

```
INamedVariable* Var()
```

#### Возвращаемое значение

Нечёткая переменная для данного условия.

### Var (Метод Set)

Устанавливает нечёткую переменную для данного условия

```
void Var(  
    INamedVariable*& value // нечёткая переменная для данного условия  
)
```

### Параметры

*value*  
[in] нечёткая переменная.

## CConditions

Класс задает набор нечетких условий, связанных оператором.

### Описание

Набор нечетких условий, связанных оператором, может быть описан, например, следующим выражением:

$$(X_1 \text{ is } a_1) \wedge (X_2 \text{ is } a_2) \wedge \dots \wedge (X_n \text{ is } a_n)$$

где:

- X = (X1, X2, X3 ... Xn) – вектор входных переменных;
- a = (a1, a2, a3 ... an) – вектор значений входных переменных.

В данном примере использован оператор *and* (и). Также в этом классе доступен оператор *or* (или).

### Декларация

```
class CConditions : public ICondition
```

### Заголовок

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

### Иерархия наследования

```
CObject
  ICondition
    CConditions
```

### Методы класса

Метод класса	Описание
<a href="#">ConditionsList</a>	Возвращает список всех условий
<a href="#">Not</a>	Возвращает и устанавливает флаг, указывающий на то, применять ли к этим условиям отрицание
<a href="#">Or</a>	Возвращает и устанавливает тип оператора связки условий.

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## ConditionsList

Возвращает список всех условий.

```
CList* ConditionsList()
```

#### Возвращаемое значение

Список всех условий.

## Not (Метод Get)

Возвращает флаг, указывающий на то, применять ли к этим условиям отрицание.

```
bool Not()
```

#### Возвращаемое значение

Значение флага.

## Not (Метод Set)

Устанавливает флаг указывающий на то, применять ли к этим условиям отрицание

```
void Not(
    bool not      // значение флага
)
```

#### Параметры

*not*

[in] Значение флага.

## Op (Метод Get)

Возвращает тип оператора связки условий. Доступны два оператора: *and* (и) и *or* (или).

```
OperatorType Op()
```

#### Возвращаемое значение

Тип оператора связки условий.

## Op (Метод Set)

Устанавливает оператор связки условий. Доступны два оператора: *and* (и) и *or* (или).

```
void Op(
    OperatorType op      // тип оператора связки условий
)
```

#### Параметры

*op*

[in] Тип оператора связки условий.



## CGenericFuzzyRule

Базовый класс для обоих типов нечетких правил.

### Декларация

```
class CGenericFuzzyRule : public IParsableRule
```

### Заголовок

```
#include <Math\Fuzzy\fuzzyrule.mqh>
```

### Иерархия наследования

[CObject](#)

IParsableRule

CGenericFuzzyRule

### Прямые потомки

[CMamdaniFuzzyRule](#), [CSugenoFuzzyRule](#)

### Методы класса

Метод класса	Описание
<a href="#">Conclusion</a>	Возвращает и устанавливает заключение нечёткого правила
<a href="#">Condition</a>	Возвращает и устанавливает условие (набор условий) 'if' для нечёткого правила
<a href="#">CreateCondition</a>	Создаёт условие для нечёткого правила по заданным параметрам

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## Conclusion (Метод Get)

Возвращает заключение нечёткого правила.

```
CSingleConditon* Conclusion()
```

### Возвращаемое значение

Заключение нечеткого правила.

## Conclusion (Метод Set)

Устанавливает заключение нечёткого правила.

```
virtual void Conclusion(
```

```
CSingleConditon* value // заключение нечёткого правила
)
```

**Параметры***value*

[in] Заключение нечёткого правила.

**Condition (Метод Get)**

Возвращает условие (набор условий) 'if' для нечёткого правила.

```
CConditons* Condition()
```

**Возвращаемое значение**

Нечёткое условие (набор условий).

**Condition (Метод Set)**

Устанавливает условие (набор условий) 'if' для нечёткого правила.

```
void Condition(
    CConditons* value // условие (набор условий) "if" для нечёткого правила
)
```

**Параметры***value*

[in] Нечёткое условие (набор условий).

**CreateCondition**

Создаёт условие для нечёткого правила по заданным параметрам.

```
CFuzzyCondition* CreateCondition(
    CFuzzyVariable* var, // нечёткая переменная
    CFuzzyTerm* term, // нечёткий терм
)
```

**Параметры***var*

[in] Нечёткая переменная.

*term*

[in] Нечёткий терм.

**Возвращаемое значение**

Состояние нечёткого правила.

## CreateCondition

Создаёт условие для нечёткого правила по заданным параметрам.

```
CFuzzyCondition* CreateCondition(
    CFuzzyVariable* var,           // нечёткая переменная
    CFuzzyTerm*      term,          // нечёткий терм
    bool            not,            // флаг, указывающий, нужно ли применять к условию отрицание
)
```

### Параметры

*var*

[in] Нечёткая переменная.

*term*

[in] Нечёткий терм.

*not*

[in] Флаг, указывающий, нужно ли применять к условию отрицание.

### Возвращаемое значение

Состояние нечёткого правила.

## CreateCondition

Создаёт условие для нечёткого правила по заданным параметрам.

```
CFuzzyCondition* CreateCondition(
    CFuzzyVariable* var,           // нечёткая переменная
    CFuzzyTerm*      term,          // нечёткий терм
    bool            not,            // флаг, указывающий, нужно ли применять к условию отрицание
    HedgeType       hedge           // тип связки условия
)
```

### Параметры

*var*

[in] Нечёткая переменная.

*term*

[in] Нечёткий терм.

*not*

[in] Флаг, указывающий, нужно ли применять к условию отрицание.

*hedge*

[in] Тип связки условия.

### Возвращаемое значение

Состояние нечёткого правила.

## Переменные для нечетких систем

В нечётких системах применяются **нечёткие (лингвистические) переменные**. Это такие переменные, значениями которых могут быть слова или словосочетания некоторого естественного или искусственного языка.

Лингвистические переменные составляют нечеткие множества. Характер и количество нечётких переменных при определении нечётких множеств изменяются для каждой отдельной задачи.

Класс	Описание
<a href="#">CFuzzyVariable</a>	Класс, с помощью которого создаются нечеткие переменные общего вида.
<a href="#">CSugenoVariable</a>	Класс, с помощью которого создаются нечеткие переменные типа Сугено.

## CFuzzyVariable

Класс, с помощью которого создаются нечеткие переменные общего вида.

### Описание

Нечеткая переменная в нашем случае задается следующими параметрами:

- максимальное значение переменной;
- минимальное значение переменной;
- имя нечеткой переменной;
- терм-множество (множество всех возможных значений, которые способна принимать лингвистическая переменная).

### Декларация

```
class CFuzzyVariable : public CNamedVariableImpl
```

### Заголовок

```
#include <Math\Fuzzy\fuzzyvariable.mqh>
```

### Иерархия наследования

```
CObject
  INamedValue
    INamedVariable
      CNamedVariableImpl
        CFuzzyVariable
```

### Методы класса

Метод класса	Описание
<a href="#"><u>AddTerm</u></a>	Добавляет один нечеткий терм к нечеткой переменной.
<a href="#"><u>GetTermByName</u></a>	Получает нечеткий терм по заданному имени.
<a href="#"><u>Max</u></a>	Возвращает и устанавливает максимальное значение для нечеткой переменной.
<a href="#"><u>Min</u></a>	Возвращает и устанавливает минимальное значение для нечеткой переменной.
<a href="#"><u>Terms</u></a>	Возвращает и устанавливает список нечетких термов для данной нечеткой переменной.
<a href="#"><u>Values</u></a>	Возвращает и устанавливает список нечетких термов для данной нечеткой переменной.

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CNamedVariableImpl

Name, Name

## AddTerm

Добавляет один нечёткий терм к нечёткой переменной.

```
void AddTerm(
    CFuzzyTerm*& term      // нечеткий терм
)
```

#### Параметры

*term*

[in] Нечёткий терм.

## GetTermByName

Получает нечёткий терм по заданному имени.

```
CFuzzyTerm* GetTermByName (
    const string name      // имя нечёткого терма
)
```

#### Параметры

*name*

[in] Имя нечёткого терма.

#### Возвращаемое значение

Нечёткий терм с заданным именем.

## Max (Метод Get)

Возвращает максимальное значение для нечёткой переменной.

```
double Max()
```

#### Возвращаемое значение

Максимальное значение для нечеткой переменной.

## Max (Метод Set)

Устанавливает максимальное значение для нечёткой переменной.

```
void Max (
    const double max      // максимальное значение для нечёткой переменной
)
```

## Параметры

*max*

[in] Максимальное значение для нечёткой переменной.

## Min (Метод Get)

Возвращает минимальное значение для нечёткой переменной.

```
double Min()
```

### Возвращаемое значение

Минимальное значение для нечеткой переменной.

## Max (Метод Set)

Устанавливает минимальное значение для нечёткой переменной.

```
void Min(  
    const double min // минимальное значение для нечёткой переменной  
)
```

## Параметры

*min*

[in] Минимальное значение для нечёткой переменной.

## Terms (Метод Get)

Возвращает список нечётких термов для данной нечёткой переменной.

```
CList* Terms()
```

### Возвращаемое значение

Список нечётких термов для данной нечёткой переменной.

## Terms (Метод Set)

Устанавливает список нечётких термов для данной нечёткой переменной.

```
void Terms(  
    CList*& terms // список нечётких термов для данной переменной  
)
```

## Параметры

*terms*

[in] список нечётких термов для данной нечёткой переменной.

## Values

Возвращает список нечётких термов для данной нечёткой переменной.

```
CList* Values()
```

#### Возвращаемое значение

Список нечётких термов для данной переменной.

## CSugenoVariable

Класс, с помощью которого создаются нечеткие переменные типа Сугено.

### Описание

Нечеткая переменная типа Сугено отличается от лингвистической переменной общего типа тем, что задается не терм-множеством, а набором линейных функций.

### Декларация

```
class CSugenoVariable : public CNamedVariableImpl
```

### Заголовок

```
#include <Math\Fuzzy\sugenovariable.mqh>
```

### Иерархия наследования

[CObject](#)

```
INamedValue
INamedVariable
CNamedVariableImpl
CSugenoVariable
```

### Методы класса

Метод класса	Описание
<a href="#"><u>Functions</u></a>	Возвращает список линейных функций нечёткой переменной Сугено.
<a href="#"><u>GetFuncByName</u></a>	Возвращает линейную функцию по заданному имени.
<a href="#"><u>Values</u></a>	Возвращает список линейных функций нечёткой переменной Сугено.

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CNamedVariableImpl

[Name](#), [Name](#)

## Functions

Возвращает список линейных функций нечёткой переменной Сугено.

```
CList* Functions()
```

### Возвращаемое значение

Список линейных функций.

## GetFuncByName

Возвращает линейную функцию по заданному имени.

```
ISugenoFunction* GetFuncByName(
    const string name // имя линейной функции
)
```

### Параметры

*name*

[in] Имя линейной функции.

### Возвращаемое значение

Линейная функция с заданным именем.

## Values

Возвращает список линейных функций нечёткой переменной Сугено.

```
CList* Values()
```

### Возвращаемое значение

Список линейных функций нечёткой переменной Сугено.

## CFuzzyTerm (нечеткие термы)

Класс для реализации нечетких термов.

### Описание

Термом (*term*) называется любой элемент терм-множества. Терм определяется двумя составляющими:

- именем нечеткого терма;
- функцией принадлежности.

### Декларация

```
class CFuzzyTerm : public CNamedValueImpl
```

### Заголовок

```
#include <Math\Fuzzy\fuzzyterm.mqh>
```

### Иерархия наследования

```
CObject
INamedValue
CNamedValueImpl
CFuzzyTerm
```

### Методы класса

Метод класса	Описание
<a href="#">MembershipFunction</a>	Возвращает функцию принадлежности для данного нечеткого терма.

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CNamedValueImpl

[Name](#), [Name](#)

## MembershipFunction

Возвращает функцию принадлежности для данного нечеткого терма.

```
IMembershipFunction* MembershipFunction()
```

### Возвращаемое значение

Функция принадлежности.

## Нечеткие системы

**Нечеткая система** (или **нечеткая модель**) — математическая модель, в основе вычисления которой лежит нечеткая логика. К построению таких моделей прибегают в случае, когда предмет исследования имеет очень слабую формализацию, и его точное математическое описание слишком сложное или неизвестно.

Ход построения модели можно разделить на три основных этапа:

1. Определение входных и выходных параметров модели.
2. Построение базы знаний.
3. Выбор одного из методов нечеткого логического вывода (Мамдани или Сугено).

От первого этапа непосредственно зависят два других, и именно он определяет будущее функционирование модели.

**База знаний** (*база правил*) — это совокупность нечетких правил вида: "если, то", определяющих взаимосвязь между входами и выходами исследуемого объекта.

**Условие** (*Condition*) правила характеризует текущее состояние объекта, а **заключение** (*Conclusion*) — то, как это условие повлияет на объект.

Условия и заключения для каждого правила могут быть двух видов:

1. простое (ссылка на `Csinglcond`) — в нем участвует одна нечеткая переменная;
2. составное (ссылка `Cconditions`) — участвуют несколько нечетких переменных.

Каждое правило в системе имеет **вес** — значимость правила в модели. Весовые коэффициенты присваиваются правилу в диапазоне [0, 1].

В зависимости от созданной базы знаний для модели определяется система нечеткого логического вывода. **Нечетким логическим выводом** называется получение заключения в виде нечеткого множества, соответствующего текущим значениям входов, с использованием нечеткой базы знаний и нечетких операций. Два основных типа нечеткого логического вывода — Мамдани и Сугено.

## Система Мамдани

Значения выходной переменной в системе Мамдани задаются нечеткими термами.

### Описание

Нечеткое логическое правило для алгоритма Мамдани можно описать следующим выражением:

$$if(X_1 \text{ is } a_1) \wedge (X_2 \text{ is } a_2) \wedge \dots \wedge (X_n \text{ is } a_n) \text{ then } (Y \text{ is } d)(W)$$

где:

- X = (X1, X2, X3 ... Xn) – вектор входных переменных;
- Y – выходная переменная;
- a = (a1, a2, a3 ... an) – вектор значений входных переменных;
- d – значение выходной переменной;
- W – вес правила.

### Методы класса

Метод класса	Описание
<a href="#">AggregationMethod</a>	Устанавливает тип агрегации условий
<a href="#">Calculate</a>	Производит расчет нечеткого логического вывода для данной системы
<a href="#">DefuzzificationMethod</a>	Устанавливает тип метода дефазификации
<a href="#">EmptyRule</a>	Создает пустое нечеткое правило Мамдани на основе текущей системы
<a href="#">ImplicationMethod</a>	Устанавливает тип оператора импликации системы
<a href="#">Output</a>	Возвращает список выходных нечетких переменных Мамдани.
<a href="#">OutputByName</a>	Возвращает выходную нечеткую переменную Мамдани по заданному имени.
<a href="#">ParseRule</a>	Создаёт нечеткое правило Мамдани на основе заданной строки.
<a href="#">Rules</a>	Возвращает список нечетких правил Мамдани.

#### Методы унаследованные от CGenericFuzzySystem

Input, AndMethod, AndMethod, OrMethod, OrMethod, InputByName, Fuzzify

## AggregationMethod

Устанавливает тип метода агрегации условий.

```
void AggregationMethod(
    AggregationMethod value          // тип метода агрегации
)
```

#### Параметры

*value*  
 [in] Тип метода агрегации условий.

## Calculate

Производит расчет нечёткого логического вывода для данной системы.

```
CList* Calculate(
    CList* inputValues           // входные данные
)
```

#### Параметры

*inputValues*  
 [in] Входные данные для расчета.

#### Возвращаемое значение

Результат расчетов.

## DefuzzificationMethod

Устанавливает тип метода дефазификации.

```
void DefuzzificationMethod(
    DefuzzificationMethod value        // тип метода дефазификации.
)
```

#### Параметры

*value*  
 [in] Тип метода дефазификации.

## EmptyRule

Создаёт пустое нечёткое правило Мамдани на основе текущей системы.

```
CMamdaniFuzzyRule* EmptyRule()
```

#### Возвращаемое значение

Нечёткое правило Мамдани.

## ImplicationMethod

Устанавливает тип оператора импликации условий.

```
void ImplicationMethod(
    ImplicationMethod value      // тип оператора импликации
)
```

#### Параметры

*value*  
 [in] Тип оператора импликации условий.

## Output

Возвращает список выходных нечётких переменных Мамдани.

```
CList* Output()
```

#### Возвращаемое значение

Список нечётких переменных.

## OutputByName

Возвращает выходную нечёткую переменную Мамдани по заданному имени.

```
CFuzzyVariable* OutputByName(
    const string name      // имя нечёткой переменной
)
```

#### Параметры

*name*  
 [in] Имя нечёткой переменной.

#### Возвращаемое значение

Нечёткая переменная Мамдани с заданным именем.

## ParseRule

Создаёт нечёткое правило Мамдани на основе заданной строки.

```
CMamdaniFuzzyRule* ParseRule(
    const string rule      // строковое представление нечёткого правила
)
```

#### Параметры

*rule*  
 [in] Строковое представление нечёткого правила Мамдани.

#### Возвращаемое значение

Нечёткое правило Мамдани.

## Rules

Возвращает список нечётких правил Мамдани.

```
CList* Rules()
```

### Возвращаемое значение

Список нечётких правил Мамдани.

## CSugenoFuzzyRule

Нечеткая логическая система Сугено – один из двух основных типов нечетких систем. Значения выходной переменной в нем задаются как линейная комбинация входных переменных.

### Описание

Отличие от системы Мамдани заключается в том, что значение выходной переменной задается не нечетким термом, а линейной функцией от входов. Нечеткое логическое правило для алгоритма Сугено можно описать следующим выражением:

$$if(X_1 \text{ is } a_1) \wedge (X_2 \text{ is } a_2) \wedge \dots \wedge (X_n \text{ is } a_n) \text{ then } (Y = b_0 + b_1 \cdot X_1 + b_2 \cdot X_2 + \dots + b_n \cdot X_n)(W)$$

где:

- X = (X1, X2, X3 ... Xn) – вектор входных переменных;
- Y – выходная переменная;
- a = (a1, a2, a3 ... an) – вектор значений входных переменных;
- b = (b1, b2, b3 ... bn) – коэффициент при свободном члене в линейной функции для выходного значения
- W – вес правила.

### Методы класса

Метод класса	Описание
<a href="#"><u>Calculate</u></a>	Производит расчет нечёткого логического вывода для данной системы.
<a href="#"><u>CreateSugenoFunction</u></a>	Создаёт линейную функцию Сугено для данной системы.
<a href="#"><u>EmptyRule</u></a>	Создаёт пустое нечёткое правило Сугено на основе текущей системы.
<a href="#"><u>Output</u></a>	Возвращает список выходных нечётких переменных Сугено.
<a href="#"><u>OutputByName</u></a>	Возвращает выходную нечёткую переменную Сугено по заданному имени.
<a href="#"><u>ParseRule</u></a>	Создаёт нечёткое правило Сугено на основе заданной строки.
<a href="#"><u>Rules</u></a>	Возвращает список нечётких правил.

### Методы унаследованные от CGenericFuzzySystem

Input, AndMethod, AndMethod, OrMethod, OrMethod, InputByName, Fuzzify

## Calculate

Производит расчет нечёткого логического вывода для данной системы.

```
CList* Calculate(
    CList*& inputValues           // входные данные
)
```

**Параметры***inputValues*

[in] Входные данные для расчета.

**Возвращаемое значение**

Результат расчетов.

## CreateSugenoFunction

Создаёт линейную функцию Сугено для данной системы.

```
CLinearSugenoFunction* CreateSugenoFunction(
    const string name,           // имя функции
    const double& coeffs[]       // коэффициенты функции
)
```

**Параметры***name*

[in] Имя функции.

*coeffs[]*

[in] Коэффициенты функции.

**Возвращаемое значение**

Линейная функция Сугено.

**Примечание**

Размер массива коэффициентов может быть либо равен количеству входных переменных, либо больше него на единицу. В первом случае свободный член линейной функции Сугено будет равен нулю, а во втором – последнему коэффициенту.

## CreateSugenoFunction

Создаёт линейную функцию Сугено для данной системы.

```
CLinearSugenoFunction* CreateSugenoFunction(
    const string name,           // имя функции
    CList*& coeffs,             // список пар нечёткая переменная – коэффициент при не
    const double constValue      // коэффициент при свободном члене функции
)
```

**Параметры***name*

[in] Имя функции.

*coeffs[]*  
 [in] Коэффициенты функции.

#### Возвращаемое значение

Линейная функция Сугено.

## EmptyRule

Создаёт пустое нечёткое правило Сугено на основе текущей системы.

```
CSugenoFuzzyRule* EmptyRule()
```

#### Возвращаемое значение

Нечёткое правило Сугено.

## Output

Возвращает список выходных нечётких переменных Сугено.

```
CList* Output()
```

#### Возвращаемое значение

Список нечётких переменных.

## OutputByName

Возвращает выходную нечёткую переменную Сугено по заданному имени.

```
CSugenoVariable* OutputByName(  
  const string name // имя нечеткой переменной  
)
```

#### Параметры

*name*  
 Имя нечёткой переменной.

#### Возвращаемое значение

Нечёткая переменная Сугено с заданным именем.

## ParseRule

Создаёт нечёткое правило Сугено на основе заданной строки.

```
CSugenoFuzzyRule* ParseRule(  
  const string rule // строковое представление нечёtkого правила Сугено  
)
```

#### Параметры

*rule*

[in] Строковое представление нечёткого правила Сугено.

#### Возвращаемое значение

Нечеткое правило Сугено.

## Rules

Возвращает список нечётких правил.

```
CList* Rules()
```

#### Возвращаемое значение

Список нечетких правил.

## Класс для работы с программами OpenCL

Класс COpenCL является оберткой для удобной работы с [функциями OpenCL](#). Использование GPU в некоторых случаях позволяет значительно увеличить скорость вычислений.

Примеры использования класса для расчетов на числах float и double можно найти в папке MQL5\Scripts\Examples\OpenCL\ в соответствующих подкаталогах, исходные коды самих OpenCL-программ находятся в подкаталогах MQL5\Scripts\Examples\OpenCL\Double\Kernels и MQL5\Scripts\Examples\OpenCL\Float\Kernels .

- MatrixMult.mq5 - пример умножения матриц с использованием глобальной и локальной памяти
- BitonicSort.mq5 - пример параллельной сортировки элементов массива на GPU
- FFT.mq5 - пример расчета быстрого преобразования Фурье
- Wavelet.mq5 - пример расчета вейвлет-преобразования данных при помощи вейвлета Морле.

Исходный код OpenCL рекомендуется писать в отдельных CL-файлах, которые затем можно подключать к MQL5-программе с помощью [ресурсных переменных](#).

### Декларация

```
class COpenCL
```

### Заголовок

```
#include <OpenCL\OpenCL.mqh>
```

### Методы класса

Имя	Описание
<a href="#">BufferCreate</a>	Создает буфер OpenCL по указанному индексу
<a href="#">BufferFree</a>	Удаляет буфер по указанному индексу
<a href="#">BufferFromArray</a>	Создает буфер по указанному индексу из массива значений
<a href="#">BufferRead</a>	Считывает в массив буфер OpenCL по указанному индексу
<a href="#">BufferWrite</a>	Записывает массив данных в буфер по указанному индексу
<a href="#">Execute</a>	Выполняет OpenCL программу по указанному индексу
<a href="#">GetContext</a>	Возвращает хендл контекста OpenCL
<a href="#">GetKernel</a>	Возвращает хендл объекта кернел по указанному индексу
<a href="#">GetKernelName</a>	Возвращает имя объекта кернел по указанному индексу

<a href="#"><u>GetProgram</u></a>	Возвращает хендл программы OpenCL
<a href="#"><u>Initialize</u></a>	Инициализирует программу OpenCL
<a href="#"><u>KernelCreate</u></a>	Создает точку входа в программу OpenCL по указанному индексу
<a href="#"><u>KernelFree</u></a>	Удаляет функцию запуска OpenCL по указанному индексу
<a href="#"><u>SetArgument</u></a>	Выставляет параметр для функции OpenCL по указанному индексу
<a href="#"><u>SetArgumentBuffer</u></a>	Выставляет буфер OpenCL в качестве параметра функции OpenCL по указанному индексу
<a href="#"><u>SetArgumentLocalMemory</u></a>	Выставляет параметр в локальной памяти для функции OpenCL по указанному индексу
<a href="#"><u>SetBuffersCount</u></a>	Устанавливает количество буферов
<a href="#"><u>SetKernelsCount</u></a>	Устанавливает количество объектов кернел
<a href="#"><u>Shutdown</u></a>	Выгружает программу OpenCL
<a href="#"><u>SupportDouble</u></a>	Узнает наличие поддержки на устройстве вещественных типов данных

## BufferCreate

Создает буфер OpenCL по указанному индексу.

```
bool BufferCreate(
    const int   buffer_index,      // индекс буфера
    const uint  size_in_bytes,     // размер буфера в байтах
    const uint  flags             // комбинация флагов, задающих свойства буфера
);
```

### Параметры

*buffer\_index*

[in] Индекс буфера.

*size\_in\_bytes*

[in] Размер буфера в байтах.

*flags*

[in] Свойства буфера, задаваемые через комбинацию флагов.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## BufferFree

Удаляет буфер по указанному индексу.

```
bool BufferFree(
    const int buffer_index           // индекс буфера
);
```

### Параметры

*buffer\_index*

[in] Индекс буфера.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## BufferFromArray

Создает буфер по указанному индексу из массива значений.

```
template<typename T>
bool BufferFromArray(
    const int    buffer_index,           // индекс буфера
    T          &data[],                // массив значений
    const uint   data_array_offset,     // смещение в массиве значений в байтах
    const uint   data_array_count,      // количество значений из массива для записи
    const uint   flags                 // комбинация флагов, задающих свойства буфера
);
```

### Параметры

*buffer\_index*

[in] Индекс буфера.

*&data[]*

[in] Массив значений, которые необходимо записать в буфер OpenCL.

*data\_array\_offset*

[in] Смещение в массиве значений в байтах, с которого начинается запись.

*data\_array\_count*

[in] Количество значений, которые нужно записать.

*flags*

[in] Свойства буфера, задаваемые через комбинацию флагов.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## BufferRead

Считывает в массив буфер OpenCL по указанному индексу.

```
template<typename T>
bool BufferRead(
    const int    buffer_index,           // индекс буфера
    T          &data[],                // массив значений
    const uint   cl_buffer_offset,       // смещение в OpenCL буфере в байтах
    const uint   data_array_offset,      // смещение в массиве в элементах
    const uint   data_array_count       // количество значений из буфера для чтения
);
```

### Параметры

*buffer\_index*

[in] Индекс буфера.

*&data[]*

[in] Массив для получения значений из буфера OpenCL.

*cl\_buffer\_offset*

[in] Смещение в OpenCL буфере в байтах, с которого начинается чтение.

*data\_array\_offset*

[in] Индекс первого элемента массива для записи значений буфера OpenCL.

*data\_array\_count*

[in] Количество значений, которые нужно считать.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## BufferWrite

Записывает массив данных в буфер по указанному индексу.

```
template<typename T>
bool BufferWrite(
    const int    buffer_index,           // индекс буфера
    T          &data[],                // массив значений
    const uint   cl_buffer_offset,       // смещение в OpenCL буфере в байтах
    const uint   data_array_offset,      // смещение в массиве в элементах
    const uint   data_array_count       // количество значений из массива для записи
);
```

### Параметры

*buffer\_index*

[in] Индекс буфера.

*&data[]*

[in] Массив значений, которые необходимо записать в буфер OpenCL.

*cl\_buffer\_offset*

[in] Смещение в OpenCL буфере в байтах, с которого начинается запись.

*data\_array\_offset*

[in] Индекс первого элемента массива, начиная с которого берутся значения из массива для записи в OpenCL буфер.

*data\_array\_count*

[in] Количество значений, которые нужно записать.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## Execute

Выполняет OpenCL программу по указанному индексу.

```
bool Execute(
    const int kernel_index,           // индекс кернела
    const int work_dim,              // размерность пространства задач
    const uint &work_offset[],       // начальное смещение в пространстве задач
    const uint &work_size[]          // общее количество задач
);
```

Выполняет OpenCL программу по указанному индексу с заданным количеством задач в локальной группе.

```
bool Execute(
    const int kernel_index,           // индекс кернела
    const int work_dim,              // размерность пространства задач
    const uint &work_offset[],       // начальное смещение в пространстве задач
    const uint &work_size[],         // общее количество задач
    const uint &local_work_size[]   // количество задач в локальной группе
);
```

### Параметры

*kernel\_index*

[in] Индекс объекта кернел.

*work\_dim*

[in] Размерность пространства задач.

*&work\_offset[]*

[in] Начальное смещение в пространстве задач.

*&work\_size[]*

[in] Размер подмножества задач.

*&local\_work\_size[]*

[in] Размер локального подмножества задач в группе.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## GetContext

Возвращает хендл контекста OpenCL.

```
int GetContext();
```

### Возвращаемое значение

Хендл контекста OpenCL.

## GetKernel

Возвращает хэндл объекта кернел по указанному индексу.

```
int GetKernel(
    const int kernel_index           // индекс кернела
);
```

### Параметры

*kernel\_index*  
[in] Индекс объекта кернел.

### Возвращаемое значение

Хэндл объекта кернел.

## GetKernelName

Возвращает имя объекта кернел по указанному индексу.

```
string GetKernelName(
    const int kernel_index           // индекс кернела
);
```

### Параметры

*kernel\_index*  
[in] Индекс объекта кернел.

### Возвращаемое значение

Имя объекта кернел.

## GetProgram

Возвращает хендл программы OpenCL.

```
int GetProgram();
```

### Возвращаемое значение

Хендл программы OpenCL.

## Initialize

Инициализирует программу OpenCL.

```
bool Initialize(
    const string program,           // хэндл программы OpenCL
    const bool    show_log=true     // вести запись в лог
);
```

### Параметры

*program*

[in] Хэндл программы OpenCL.

*show\_log=true*

[in] Включить запись сообщений в журнал.

### Возвращаемое значение

Возвращает true, если инициализация прошла успешно. В противном случае возвращает false.

## KernelCreate

Создает точку входа в программу OpenCL по указанному индексу.

```
bool KernelCreate(
    const int     kernel_index,      // индекс кернела
    const string  kernel_name       // имя кернела
);
```

### Параметры

*kernel\_index*

[in] Индекс объекта кернел.

*kernel\_name*

[in] Имя объекта кернел.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## KernelFree

Удаляет функцию запуска OpenCL по указанному индексу.

```
bool KernelFree(
    const int kernel_index           // индекс кернела
);
```

### Параметры

*kernel\_index*  
[in] Индекс объекта кернел.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## SetArgument

Выставляет параметр для функции OpenCL по указанному индексу.

```
template<typename T>
bool SetArgument(
    const int kernel_index,           // индекс кернела
    const int arg_index,              // индекс аргумента функции
    T      value                    // исходный код
);
```

### Параметры

*kernel\_index*

[in] Индекс объекта кернел.

*arg\_index*

[in] Индекс аргумента функции.

*value*

[in] Значение аргумента функции.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## SetArgumentBuffer

Выставляет буфер OpenCL в качестве параметра функции OpenCL по указанному индексу.

```
bool SetArgumentBuffer(
    const int kernel_index,           // индекс кернела
    const int arg_index,              // индекс аргумента функции
    const int buffer_index           // индекс буфера
);
```

### Параметры

*kernel\_index*

[in] Индекс объекта кернел.

*arg\_index*

[in] Индекс аргумента функции.

*buffer\_index*

[in] Индекс буфера.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## SetArgumentLocalMemory

Выставляет параметр в локальной памяти для функции OpenCL по указанному индексу.

```
bool SetArgumentLocalMemory(
    const int kernel_index,           // индекс кернела
    const int arg_index,              // индекс аргумента функции
    const int local_memory_size       // размер локальной памяти
);
```

### Параметры

*kernel\_index*

[in] Индекс объекта кернел.

*arg\_index*

[in] Индекс аргумента функции.

*local\_memory\_size*

[in] Размер локальной памяти.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## SetBuffersCount

Устанавливает количество буферов.

```
bool SetBuffersCount(
    const int total_buffers // количество буферов
);
```

### Параметры

*total\_buffers*

[in] Общее количество буферов.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## SetKernelsCount

Устанавливает количество объектов кернел.

```
bool SetKernelsCount(
    const int total_kernels           // количество кернелов
);
```

### Параметры

*total\_kernels*

[in] Общее количество кернелов.

### Возвращаемое значение

В случае успешного выполнения возвращает true, в противном случае false.

## Shutdown

Выгружает программу OpenCL.

```
void Shutdown();
```

### Возвращаемое значение

Нет возвращаемых значений.

## SupportDouble

Узнает наличие поддержки на устройстве вещественных типов данных.

```
bool SupportDouble();
```

### Возвращаемое значение

Возвращает true, если на устройстве поддерживаются вещественные типы данных.

## Базовый класс CObject

Класс CObject является базовым классом для построения стандартной библиотеки MQL5.

### Описание

Класс CObject обеспечивает всем своим потомкам возможность быть элементом связанного списка. Кроме того определяется ряд виртуальных методов для дальнейшей реализации в классах-потомках.

### Декларация

```
class CObject
```

### Заголовок

```
#include <Object.mqh>
```

### Иерархия наследования

CObject

#### Прямые потомки

[CAccountInfo](#), [CArray](#), [CChart](#), [CChartData](#), [CCurve](#), [CDealInfo](#), [CDictionary\\_Obj\\_Double](#), [CDictionary\\_Obj\\_Obj](#), [CDictionary\\_String\\_Obj](#), [CExpertBase](#), [CFile](#), [CHistoryOrderInfo](#), [CList](#), [COrderInfo](#), [CPositionInfo](#), [CString](#), [CSymbolInfo](#), [CTerminalInfo](#), [CTrade](#), [CTreeNode](#), [CWnd](#), [ICondition](#), [IExpression](#), [IMembershipFunction](#), [INamedValue](#), [IParsableRule](#)

### Методы класса по группам

Атрибуты	
<a href="#">Prev</a>	Получает значение предыдущего элемента
<a href="#">Prev</a>	Устанавливает значение предыдущего элемента
<a href="#">Next</a>	Получает значение последующего элемента
<a href="#">Next</a>	Устанавливает значение последующего элемента
Сравнение	
virtual <a href="#">Compare</a>	Возвращает результат сравнения с другим объектом
Ввод/вывод	
virtual <a href="#">Save</a>	Записывает объект в файл
virtual <a href="#">Load</a>	Читает объект из файла
virtual <a href="#">Type</a>	Возвращает тип объекта



## Prev

Получает указатель на предыдущий элемент списка.

```
CObject* Prev()
```

### Возвращаемое значение

Указатель на предыдущий элемент списка. Если элемент в списке первый, то возвращается NULL.

### Пример:

```
//--- example for CObject::Prev()
#include <Object.mqh>
//---
void OnStart()
{
    CObject *object_first,*object_second;
    //---
    object_first=new CObject;
    if(object_first==NULL)
    {
        printf("Object create error");
        return;
    }
    object_second=new CObject;
    if(object_second==NULL)
    {
        printf("Object create error");
        delete object_first;
        return;
    }
    //--- set interconnect
    object_first.Next(object_second);
    object_second.Prev(object_first);
    //--- use prev object
    CObject *object=object_second.Prev();
    //--- delete objects
    delete object_first;
    delete object_second;
}
```

## Prev

Устанавливает указатель на предыдущий элемент списка.

```
void Prev(
    CObject* object      // указатель на предыдущий элемент списка
)
```

### Параметры

*object*

[in] Новое значение указателя на предыдущий элемент списка.

### Пример:

```
//--- example for CObject::Prev(CObject*)
#include <Object.mqh>
//---
void OnStart()
{
    CObject *object_first,*object_second;
//---
object_first=new CObject;
if(object_first==NULL)
{
    printf("Object create error");
    return;
}
object_second=new CObject;
if(object_second==NULL)
{
    printf("Object create error");
    delete object_first;
    return;
}
//--- set interconnect
object_first.Next(object_second);
object_second.Prev(object_first);
//--- use objects
//--- ...
//--- delete objects
delete object_first;
delete object_second;
}
```

## Next

Получает указатель на следующий элемент списка.

```
CObject* Next()
```

### Возвращаемое значение

Указатель на следующий элемент списка. Если элемент в списке последний, то возвращается NULL.

### Пример:

```
//--- example for CObject::Next()
#include <Object.mqh>
//---
void OnStart()
{
    CObject *object_first,*object_second;
//---
    object_first=new CObject;
    if(object_first==NULL)
    {
        printf("Object create error");
        return;
    }
    object_second=new CObject;
    if(object_second==NULL)
    {
        printf("Object create error");
        delete object_first;
        return;
    }
//--- set interconnect
    object_first.Next(object_second);
    object_second.Prev(object_first);
//--- use next object
    CObject *object=object_first.Next();
//--- delete objects
    delete object_first;
    delete object_second;
}
```

## Next

Устанавливает указатель на следующий элемент списка.

```
void Next(
    CObject* object      // указатель на следующий элемент списка
)
```

### Параметры

*object*

[in] Новое значение указателя на следующий элемент списка.

### Пример:

```
//--- example for CObject::Next (CObject*)
#include <Object.mqh>
//---
void OnStart()
{
    CObject *object_first,*object_second;
//---
object_first=new CObject;
if(object_first==NULL)
{
    printf("Object create error");
    return;
}
object_second=new CObject;
if(object_second==NULL)
{
    printf("Object create error");
    delete object_first;
    return;
}
//--- set interconnect
object_first.Next(object_second);
object_second.Prev(object_first);
//--- use objects
//--- ...
//--- delete objects
delete object_first;
delete object_second;
}
```

## Compare

Сравнивает данные элемента списка с данными другого элемента списка.

```
virtual int Compare(
    const CObject* node,           // элемент
    const int      mode=0          // вариант
) const
```

### Параметры

*node*  
 [in] Указатель на элемент списка для сравнения  
*mode=0*  
 [in] Вариант сравнения

### Возвращаемое значение

0 - в случае элементы списка равны, -1 - в случае если элемент списка меньше чем элемент списка для сравнения (*node*), 1 - в случае если элемент списка больше чем элемент списка для сравнения (*node*).

### Примечание

Метод Compare() в классе CObject всегда возвращает 0 и не выполняет каких-либо действий. При необходимости сравнения данных класса-наследника, метод Compare(...) должен быть реализован. Параметр mode нужно использовать при реализации многовариантного сравнения.

### Пример:

```
//--- example for CObject::Compare(...)
#include <Object.mqh>
//---
void OnStart()
{
    CObject *object_first,*object_second;
    //---
    object_first=new CObject;
    if(object_first==NULL)
    {
        printf("Object create error");
        return;
    }
    object_second=new CObject;
    if(object_second==NULL)
    {
        printf("Object create error");
        delete object_first;
        return;
    }
}
```

```
//--- set interconnect
object_first.Next(object_second);
object_second.Prev(object_first);
//--- compare objects
int result=object_first.Compare(object_second);
//--- delete objects
delete object_first;
delete object_second;
}
```

## Save

Сохраняет данные элемента списка в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого при помощи функции FileOpen() бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Примечание

Метод Save(int) в классе CObject всегда возвращает true и не выполняет каких-либо действий. При необходимости сохранения данных класса-наследника в файле, метод Save(int) должен быть реализован.

### Пример:

```
//--- example for CObject::Save(int)
#include <Object.mqh>
//---
void OnStart()
{
    int file_handle;
    CObject *object=new CObject;
    //---
    if(object!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- set objects data
    //--- . . .
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!object.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete object;
            FileClose(file_handle);
        }
    }
}
```

```
// ---
return;
}
FileClose(file_handle);
}
delete object;
}
```

## Load

Загружает данные элемента списка из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого при помощи функции FileOpen() бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Примечание

Метод Load(int) в классе CObject всегда возвращает true и не выполняет каких-либо действий. При необходимости загрузки данных класса-наследника из файла, метод Load(int) должен быть реализован.

### Пример:

```
//--- example for CObject::Load(int)
#include <Object.mqh>
//---

void OnStart()
{
    int file_handle;
    CObject *object=new CObject;
    //---

    if(object!=NULL)
    {
        printf("Object create error");
        return;
    }

    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!object.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete object;
            FileClose(file_handle);
            //---
        }
    }
}
```

```
    }
    FileClose(file_handle);
}
//--- use object
//--- . . .
delete object;
}
```

## Type

Получает идентификатор типа.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа (для CObject - 0).

### Пример:

```
//--- example for CObject::Type()
#include <Object.mqh>
//---
void OnStart()
{
    CObject *object=new CObject;
    //---
    object=new CObject;
    if(object ==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get objects type
    int type=object.Type();
    //--- delete object
    delete object;
}
```

## Структуры данных

Этот раздел содержит технические детали работы с различными структурами данных (массивами, связанными списками и др.) и описание соответствующих компонентов стандартной библиотеки MQL5.

Использование классов структур данных позволит сэкономить время при создании пользовательских хранилищ данных разнообразных форматов (в том числе составных структур данных).

Стандартная библиотека MQL5 (в части наборов данных) размещается в рабочем каталоге терминала в папке `Include\Arrays`

## Массивы данных

Использование классов динамических массивов данных позволит сэкономить время при создании пользовательских хранилищ данных разнообразных форматов (в том числе многомерных массивов).

Стандартная библиотека MQL5 (в части массивов данных) размещается в рабочем каталоге терминала в папке `Include\Arrays`.

Класс	Описание
<a href="#">CArray</a>	Базовый класс динамического массива данных
<a href="#">CArrayChar</a>	Динамический массив переменных типа <code>char</code> или <code>uchar</code>
<a href="#">CArrayShort</a>	Динамический массив переменных типа <code>short</code> или <code>ushort</code>
<a href="#">CArrayInt</a>	Динамический массив переменных типа <code>int</code> или <code>uint</code>
<a href="#">CArrayLong</a>	Динамический массив переменных типа <code>long</code> или <code>ulong</code>
<a href="#">CArrayFloat</a>	Динамический массив переменных типа <code>float</code>
<a href="#">CArrayDouble</a>	Динамический массив переменных типа <code>double</code>
<a href="#">CArrayString</a>	Динамический массив переменных типа <code>string</code>
<a href="#">CArrayObj</a>	Динамический массив указателей <code>CObject</code>
<a href="#">CList</a>	Обеспечивает возможность работы со списком экземпляров класса <a href="#">CObject</a> и его наследников

<a href="#"><u>CTreeNode</u></a>	Обеспечивает возможность работы с узлами двоичного дерева <a href="#"><u>CTree</u></a>
<a href="#"><u>CTree</u></a>	Обеспечивает возможность работы с двоичным деревом экземпляров класса <a href="#"><u>CTreeNode</u></a> и его наследников.

## CArray

Класс CArray является базовым классом динамического массива переменных.

### Описание

Класс CArray предназначен для работы с динамическим массивом переменных: распределение памяти, сортировка и работа с файлами.

### Декларация

```
class CArray : public CObject
```

### Заголовок

```
#include <Arrays\Array.mqh>
```

### Иерархия наследования

[CObject](#)

CArray

### Прямые потомки

[CArrayChar](#), [CArrayDouble](#), [CArrayFloat](#), [CArrayInt](#), [CArrayLong](#), [CArrayObj](#), [CArrayShort](#), [CArrayString](#)

### Методы класса по группам

Атрибуты	
<a href="#">Step</a>	Получает шаг приращения размеров массива
<a href="#">Step</a>	Устанавливает шаг приращения размеров массива
<a href="#">Total</a>	Получает количество элементов в массиве
<a href="#">Available</a>	Получает количество свободных элементов массива, доступных без дополнительного распределения памяти
<a href="#">Max</a>	Получает максимально возможный размер массива без перевыделения памяти
<a href="#">IsSorted</a>	Получает признак сортированности массива по указанному варианту
<a href="#">SortMode</a>	Получает вариант сортировки массива
<b>Метод очистки</b>	
<a href="#">Clear</a>	Удаляет все элементы массива без освобождения памяти

Сортировка	
<a href="#"><u>Sort</u></a>	Сортирует массив по указанному варианту
Ввод/вывод	
<a href="#"><u>virtual Save</u></a>	Сохраняет данные массива в файле
<a href="#"><u>virtual Load</u></a>	Загружает данные массива из файла

Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)

## Step

Получает шаг приращения размеров массива.

```
int Step() const
```

### Возвращаемое значение

Шаг приращения размеров массива.

### Пример:

```
//--- example for CArray::Step()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get resize step
    int step=array.Step();
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```

## Step

Устанавливает шаг приращения размеров массива.

```
bool Step(  
    int step // шаг  
)
```

### Параметры

*step*

[in] Новое значение шага приращения размеров массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка установить шаг меньше или равный нулю.

### Пример:

```
//--- example for CArray::Step(int)  
#include <Arrays\Array.mqh>  
//---  
void OnStart()  
{  
    CArray *array=new CArray;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- set resize step  
    bool result=array.Step(1024);  
    //--- use array  
    //--- ...  
    //--- delete array  
    delete array;  
}
```

## Total

Получает количество элементов в массиве.

```
int Total() const;
```

### Возвращаемое значение

Количество элементов в массиве.

### Пример:

```
//--- example for CArray::Total()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check total
    int total=array.Total();
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```

## Available

Получает количество свободных элементов массива, доступных без дополнительного распределения памяти.

```
int Available() const
```

### Возвращаемое значение

Количество свободных элементов массива, доступных без дополнительного распределения памяти.

### Пример:

```
//--- example for CArray::Available()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check available
    int available=array.Available();
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```

## Max

Получает максимально возможный размер массива без перевыделения памяти.

```
int Max() const
```

### Возвращаемое значение

Максимально возможный размер массива без перевыделения памяти.

### Пример:

```
//--- example for CArray::Max()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check maximum size
    int max=array.Max();
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```

## IsSorted

Получает признак сортированности массива по указанному варианту.

```
bool IsSorted(
    int mode=0           // вариант сортировки
) const
```

### Параметры

*mode=0*

[in] Проверяемый вариант сортировки.

### Возвращаемое значение

Флаг сортированности списка. Если список сортированный по указанному варианту – true, иначе – false.

### Примечание

Признак сортированности массива нельзя изменить непосредственно. Признак сортированности устанавливается методом Sort() и сбрасывается любыми методами добавления/вставки кроме InsertSort(...).

### Пример:

```
//--- example for CArray::IsSorted()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check sorted
    if(array.IsSorted())
    {
        //--- use methods for sorted array
        //--- ...
    }
    //--- delete array
    delete array;
}
```

## SortMode

Получает вариант сортировки массива

```
int SortMode() const;
```

### Возвращаемое значение

Вариант сортировки

### Пример:

```
//--- example for CArray::SortMode()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check sort mode
    int sort_mode=array.SortMode();
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```

## Clear

Удаляет все элементы массива без освобождения памяти.

```
void Clear()
```

### Возвращаемое значение

Нет.

### Пример:

```
//--- example for CArray::Clear()
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
//---
if(array==NULL)
{
    printf("Object create error");
    return;
}
//--- use array
//--- ...
//--- clear array
array.Clear();
//--- delete array
delete array;
}
```

## Sort

Сортирует массив по указанному варианту.

```
void Sort(
    int mode=0           // вариант сортировки
)
```

### Параметры

*mode=0*

[in] Вариант сортировки массива.

### Возвращаемое значение

Нет.

### Примечание

Сортировка массива производится всегда по возрастанию. Для массивов простых типов данных (CArrayChar, CArrayShort и др.) параметр mode не используется. Для массива CArrayObj мультивариантные сортировки должны быть реализованы в методе Sort(int) классов-наследников.

### Пример:

```
//--- example for CArray::Sort(int)
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    CArray *array=new CArray;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- sorting by mode 0
    array.Sort(0);
    //--- use array
    //--- ...
    //--- delete array
    delete array;
}
```

## Save

Сохраняет данные массива в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого при помощи функции FileOpen(...) бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArray::Save(int)
#include <Arrays\Array.mqh>
//---

void OnStart()
{
    int file_handle;
    CArray *array=new CArray;
    //---

    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- delete array
    delete array;
```

{}

## Load

Загружает данные массива из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого при помощи функции FileOpen(...) бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArray::Load(...)
#include <Arrays\Array.mqh>
//---
void OnStart()
{
    int file_handle;
    CArray *array=new CArray;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- delete array
    delete array;
```

{}

## CArrayChar

Класс CArrayChar является классом динамического массива переменных типа char или uchar.

### Описание

Класс CArrayChar обеспечивает возможность работы с динамическим массивом переменных типа char или uchar. В классе реализованы возможности добавления/вставки/удаления элементов массива, сортировки массива, поисков в отсортированном массиве. Кроме того, реализованы методы работы с файлом.

### Декларация

```
class CArrayChar : public CArray
```

### Заголовок

```
#include <Arrays\ArrayChar.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayChar
```

### Методы класса по группам

Управление памятью	
<a href="#"><u>Reserve</u></a>	Резервирует память для увеличения размеров массива
<a href="#"><u>Resize</u></a>	Устанавливает новый (меньший) размер массива
<a href="#"><u>Shutdown</u></a>	Очищает массив с полным освобождением памяти
<b>Наполнение</b>	
<a href="#"><u>Add</u></a>	Добавляет элемент в конец массива
<a href="#"><u>AddArray</u></a>	Добавляет в конец массива элементы из другого массива
<a href="#"><u>AddArray</u></a>	Добавляет в конец массива элементы из другого массива
<a href="#"><u>Insert</u></a>	Вставляет элемент в массив в указанную позицию
<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции

<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<b>Изменение</b>	
<a href="#"><u>Update</u></a>	Изменяет элемент в указанной позиции массива
<a href="#"><u>Shift</u></a>	Перемещает элемент из указанной позиции массива на указанное смещение
<b>Удаление</b>	
<a href="#"><u>Delete</u></a>	Удаляет элемент из указанной позиции массива
<a href="#"><u>DeleteRange</u></a>	Удаляет группу элементов из указанной позиции массива
<b>Доступ</b>	
<a href="#"><u>At</u></a>	Получает элемент из указанной позиции массива
<b>Сравнение</b>	
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<b>Операции с отсортированным массивом</b>	
<a href="#"><u>InsertSort</u></a>	Вставляет элемент в отсортированный массив
<a href="#"><u>Search</u></a>	Ищет элемент равный образцу в отсортированном массиве
<a href="#"><u>SearchGreat</u></a>	Ищет элемент больше образца в отсортированном массиве
<a href="#"><u>SearchLess</u></a>	Ищет элемент меньше образца в отсортированном массиве
<a href="#"><u>SearchGreatOrEqual</u></a>	Ищет элемент больше или равный образцу в отсортированном массиве
<a href="#"><u>SearchLessOrEqual</u></a>	Ищет элемент меньше или равный образцу в отсортированном массиве
<a href="#"><u>SearchFirst</u></a>	Ищет первый элемент равный образцу в отсортированном массиве

<a href="#">SearchLast</a>	Ищет последний элемент равный образцу в сортированном массиве
<a href="#">SearchLinear</a>	Ищет элемент равный образцу в массиве
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Save</a>	Сохраняет данные массива в файле
<a href="#">virtual Load</a>	Загружает данные массива из файла
<a href="#">virtual Type</a>	Получает идентификатор типа массива

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

## Reserve

Резервирует память для увеличения размеров массива.

```
bool Reserve(
    int size        // количество
)
```

### Параметры

*size*

[in] Количество дополнительных элементов массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка запросить количество меньше или равное нулю, либо если массив увеличить не удалось.

### Примечание

Для уменьшения фрагментации памяти, приращение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayChar::Reserve(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## Resize

Устанавливает новый (меньший) размер массива.

```
bool Resize(
    int size        // размер
)
```

### Параметры

*size*

[in] Новый размер массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка установить размер меньше нуля.

### Примечание

Изменение размера массива позволяет оптимально использовать память. Лишние элементы справа теряются. Для уменьшения фрагментации памяти, изменение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayChar::Resize(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- resize array
    if(!array.Resize(10))
    {
        printf("Resize error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shutdown

Очищает массив с полным освобождением памяти.

```
bool Shutdown()
```

### Возвращаемое значение

true в случае успеха, false - если произошла ошибка.

### Пример:

```
//--- example for CArrayChar::Shutdown()
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Add

Добавляет элемент в конец массива.

```
bool Add(
    char element      // элемент для добавления
)
```

### Параметры

*element*

[in] Значение элемента для добавления в массив.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элемент.

### Пример:

```
//--- example for CArrayChar::Add(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    for(int i=0;i<100;i++)
    {
        if(!array.Add(i))
        {
            printf("Element addition error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const char& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayChar::AddArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const CArrayChar* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayChar-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayChar::AddArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---

void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---

    if(array==NULL)
    {
        printf("Object create error");
        return;
    }

    //--- create source array
    CArrayChar *src=new CArrayChar;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }

    //--- add source arrays elements
    //--- . . .
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Insert

Вставляет элемент в массив в указанную позицию.

```
bool Insert(
    char element,      // элемент для вставки
    int pos            // позиция
)
```

### Параметры

*element*

[in] Значение элемента для вставки в массив

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayChar::Insert(char,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert elements
    for(int i=0;i<100;i++)
    {
        if(!array.Insert(i,0))
        {
            printf("Insert error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    const char& src[],      // массив-источник
    int          pos        // позиция
)
```

### Параметры

*src[]*  
 [in] Ссылка на массив-источник элементов для вставки

*pos*  
 [in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayChar::InsertArray(const char &[],int)
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```



## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    CArrayChar* src,          // указатель на источник
    int pos                  // позиция
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayChar-источник элементов для вставки.

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayChar::InsertArray(const CArrayChar*,int)
#include <Arrays\ArrayChar.mqh>
//---

void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayChar *src=new CArrayChar;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete src;
    }
}
```

```
    delete array;
    return;
}
//--- delete source array
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const char& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayChar::AssignArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---

void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const CArrayChar* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayChar-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayChar::AssignArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---

void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayChar *src =new CArrayChar;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- arrays is identical  
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Update

Изменяет элемент в указанной позиции массива.

```
bool Update(
    int pos,           // позиция
    char element       // значение
)
```

### Параметры

*pos*  
 [in] Позиция элемента в массиве для изменения

*element*  
 [in] Новое значение элемента

### Возвращаемое значение

true в случае успеха, false - если нет возможности изменить элемент.

### Пример:

```
//--- example for CArrayChar::Update(int,char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- update element
    if(!array.Update(0,'A'))
    {
        printf("Update error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shift

Перемещает элемент из указанной позиции массива на указанное смещение.

```
bool Shift(
    int pos,           // позиция
    int shift          // смещение
)
```

### Параметры

*pos*

[in] Позиция перемещаемого элемента массиве

*shift*

[in] Значение смещения (как положительное, так и отрицательное значение).

### Возвращаемое значение

true в случае успеха, false - если нет возможности переместить элемент.

### Пример:

```
//--- example for CArrayChar::Shift(int,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shift element
    if(!array.Shift(10,-5))
    {
        printf("Shift error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Delete

Удаляет элемент из указанной позиции массива.

```
bool Delete(  
    int pos        // позиция  
)
```

### Параметры

*pos*

[in] Позиция удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элемент.

### Пример:

```
//--- example for CArrayChar::Delete(int)  
#include <Arrays\ArrayChar.mqh>  
//---  
void OnStart()  
{  
    CArrayChar *array=new CArrayChar;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```

## DeleteRange

Удаляет группу элементов из указанной позиции массива.

```
bool DeleteRange(
    int   from,      // позиция первого элемента
    int   to        // позиция последнего элемента
)
```

### Параметры

*from*

[in] Позиция первого удаляемого элемента в массиве.

*to*

[in] Позиция последнего удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элементы.

### Пример:

```
//--- example for CArrayChar::DeleteRange(int,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- delete elements
    if(!array.DeleteRange(0,10))
    {
        printf("Delete error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## At

Получает элемент из указанной позиции массива.

```
char At(
    int pos      // позиция
) const
```

### Параметры

*pos*

[in] Позиция искомого элемента в массиве.

### Возвращаемое значение

Значение элемента в случае успеха, CHAR\_MAX- если была попытка получить элемент из не существующей позиции (при этом код последней ошибки ERR\_OUT\_OF\_RANGE).

### Примечание

Разумеется, CHAR\_MAX может быть и валидным значением элемента массива, поэтому, получив такое значение, всегда проверяйте код последней ошибки.

### Пример:

```
//--- example for CArrayChar::At(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    for(int i=0;i<array.Total();i++)
    {
        char result=array.At(i);
        if(result==CHAR_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
            //--- ошибка чтения из массива
            printf("Get element error");
            delete array;
            return;
        }
    }
    //--- use element
```

```
//--- . . .
}
//--- delete array
delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const char& src[]          // массив-источник
) const
```

### Параметры

*src []*  
[in] Ссылка на массив-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayChar::CompareArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete array
    delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const CArrayChar* src          // указатель на источник
) const
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayChar-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayChar::CompareArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayChar *src=new CArrayChar;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete arrays
    delete src;
    delete array;
}
```

## InsertSort

Вставляет элемент в сортированный массив.

```
bool InsertSort(
    char element      // элемент для вставки
)
```

### Параметры

*element*

[in] Значение элемента для вставки в сортированный массив

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayChar::InsertSort(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- insert element
    if(!array.InsertSort('A'))
    {
        printf("Insert error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Search

Ищет элемент равный образцу в сортированном массиве.

```
int Search(
    char element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayChar::Search(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.Search('A')!=-1) printf("Element found");
    else                      printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreat

Ищет элемент больше образца в сортированном массиве.

```
int SearchGreat(
    char element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayChar::SearchGreat(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreat('A')!=-1) printf("Element found");
    else                           printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLess

Ищет элемент меньше образца в сортированном массиве.

```
int SearchLess(
    char element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayChar::SearchLess(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLess('A')!=-1) printf("Element found");
    else                           printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreatOrEqual

Ищет элемент больше или равный образцу в сортированном массиве.

```
int SearchGreatOrEqual(
    char element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayChar::SearchGreatOrEqual(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreatOrEqual('A') !=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLessOrEqual

Ищет элемент меньше или равный образцу в сортированном массиве.

```
int SearchLessOrEqual(
    char element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayChar::SearchLessOrEqual(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLessOrEqual('A') !=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchFirst

Ищет первый элемент равный образцу в сортированном массиве.

```
int SearchFirst(
    char element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayChar::SearchFirst(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchFirst('A')!=-1) printf("Element found");
    else                           printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLast

Ищет последний элемент равный образцу в сортированном массиве.

```
int SearchLast(
    char element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayChar::SearchLast(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLast('A')!=-1) printf("Element found");
    else                           printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLinear

Ищет элемент равный образцу в массиве.

```
int SearchLinear(
    char element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Примечание

Поиск производится последовательным перебором элементов массива (линейный поиск для несортированных массивов).

### Пример:

```
//--- example for CArrayChar::SearchLinear(char)
#include <Arrays\ArrayChar.mqh>
//---

void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- search element
    if(array.SearchLinear('A')!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## Save

Сохраняет данные массива в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayChar::Save(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    int         file_handle;
    CArrayChar *array=new CArrayChar;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- delete array
    delete array;
```

{

## Load

Загружает данные массива из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayChar::Load(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    int file_handle;
    CArrayChar *array=new CArrayChar;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- use arrays elements
    for(int i=0;i<array.Total();i++)
```

```
{  
    printf("Element[%d] = '%c'", i, array.At(i));  
}  
//--- delete array  
delete array;  
}
```

## Type

Получает идентификатор типа массива.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа массива (для CArrayChar - 77).

### Пример:

```
//--- example for CArrayChar::Type()
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
{
    CArrayChar *array=new CArrayChar;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```

## CArrayShort

Класс CArrayShort является классом динамического массива переменных типа short или ushort.

### Описание

Класс CArrayShort обеспечивает возможность работы с динамическим массивом переменных типа short или ushort. В классе реализованы возможности добавления/вставки/удаления элементов массива, сортировки массива, поисков в отсортированном массиве. Кроме того, реализованы методы работы с файлом.

### Декларация

```
class CArrayShort : public CArray
```

### Заголовок

```
#include <Arrays\ArrayShort.mqh>
```

### Иерархия наследования

```
CObject  
CArray  
CArrayShort
```

### Методы класса по группам

Управление памятью	
<a href="#"><u>Reserve</u></a>	Резервирует память для увеличения размеров массива
<a href="#"><u>Resize</u></a>	Устанавливает новый (меньший) размер массива
<a href="#"><u>Shutdown</u></a>	Очищает массив с полным освобождением памяти
<b>Наполнение</b>	
<a href="#"><u>Add</u></a>	Добавляет элемент в конец массива
<a href="#"><u>AddArray</u></a>	Добавляет в конец массива элементы из другого массива
<a href="#"><u>AddArray</u></a>	Добавляет в конец массива элементы из другого массива
<a href="#"><u>Insert</u></a>	Вставляет элемент в массив в указанную позицию
<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции

<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<b>Изменение</b>	
<a href="#"><u>Update</u></a>	Изменяет элемент в указанной позиции массива
<a href="#"><u>Shift</u></a>	Перемещает элемент из указанной позиции массива на указанное смещение
<b>Удаление</b>	
<a href="#"><u>Delete</u></a>	Удаляет элемент из указанной позиции массива
<a href="#"><u>DeleteRange</u></a>	Удаляет группу элементов из указанной позиции массива
<b>Доступ</b>	
<a href="#"><u>At</u></a>	Получает элемент из указанной позиции массива
<b>Сравнение</b>	
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<b>Операции с отсортированным массивом</b>	
<a href="#"><u>InsertSort</u></a>	Вставляет элемент в отсортированный массив
<a href="#"><u>Search</u></a>	Ищет элемент равный образцу в отсортированном массиве
<a href="#"><u>SearchGreat</u></a>	Ищет элемент больше образца в отсортированном массиве
<a href="#"><u>SearchLess</u></a>	Ищет элемент меньше образца в отсортированном массиве
<a href="#"><u>SearchGreatOrEqual</u></a>	Ищет элемент больше или равный образцу в отсортированном массиве
<a href="#"><u>SearchLessOrEqual</u></a>	Ищет элемент меньше или равный образцу в отсортированном массиве
<a href="#"><u>SearchFirst</u></a>	Ищет первый элемент равный образцу в отсортированном массиве

<a href="#">SearchLast</a>	Ищет последний элемент равный образцу в сортированном массиве
<a href="#">SearchLinear</a>	Ищет элемент равный образцу в массиве
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Save</a>	Сохраняет данные массива в файле
<a href="#">virtual Load</a>	Загружает данные массива из файла
<a href="#">virtual Type</a>	Получает идентификатор типа массива

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

## Reserve

Резервирует память для увеличения размеров массива.

```
bool Reserve(
    int size        // количество
)
```

### Параметры

*size*

[in] Количество дополнительных элементов массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка запросить количество меньше или равное нулю, либо если массив увеличить не удалось.

### Примечание

Для уменьшения фрагментации памяти, приращение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayShort::Reserve(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## Resize

Устанавливает новый (меньший) размер массива.

```
bool Resize(
    int size        // размер
)
```

### Параметры

*size*

[in] Новый размер массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка установить размер меньше нуля.

### Примечание

Изменение размера массива позволяет оптимально использовать память. Лишние элементы справа теряются. Для уменьшения фрагментации памяти, изменение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayShort::Resize(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- resize array
    if(!array.Resize(10))
    {
        printf("Resize error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shutdown

Очищает массив с полным освобождением памяти.

```
bool Shutdown()
```

### Возвращаемое значение

true в случае успеха, false - если произошла ошибка.

### Пример:

```
//--- example for CArrayShort::Shutdown()
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Add

Добавляет элемент в конец массива.

```
bool Add(
    short element      // элемент для добавления
)
```

### Параметры

*element*

[in] Значение элемента для добавления в массив.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элемент.

### Пример:

```
//--- example for CArrayShort::Add(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    for(int i=0;i<100;i++)
    {
        if(!array.Add(i))
        {
            printf("Element addition error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const short& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayShort::AddArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const CArrayShort* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayShort-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayShort::AddArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayShort *src=new CArrayShort;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Insert

Вставляет элемент в массив в указанную позицию.

```
bool Insert(
    short element,      // элемент для вставки
    int pos            // позиция
)
```

### Параметры

*element*

[in] Значение элемента для вставки в массив

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayShort::Insert(short,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert elements
    for(int i=0;i<100;i++)
    {
        if(!array.Insert(i,0))
        {
            printf("Insert error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    const short& src[],      // массив-источник
    int           pos        // позиция
)
```

### Параметры

*src[]*  
 [in] ссылка на массив-источник элементов для вставки

*pos*  
 [in] позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayShort::InsertArray(const short &[],int)
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    CArrayShort* src,      // указатель на источник
    int          pos       // позиция
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayShort-источник элементов для вставки.

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayShort::InsertArray(const CArrayShort*,int)
#include <Arrays\ArrayShort.mqh>
//---

void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayShort *src=new CArrayShort;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete src;
    }
}
```

```
    delete array;
    return;
}
//--- delete source array
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const short& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayShort::AssignArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---

void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const CArrayShort* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayShort-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayShort::AssignArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---

void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---

    if(array==NULL)
    {
        printf("Object create error");
        return;
    }

    //--- create source array
    CArrayShort *src =new CArrayShort;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }

    //--- add source arrays elements
    //--- . . .
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- arrays is identical  
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Update

Изменяет элемент в указанной позиции массива.

```
bool Update(
    int pos,           // позиция
    short element      // значение
)
```

### Параметры

*pos*  
 [in] Позиция элемента в массиве для изменения

*element*  
 [in] Новое значение элемента

### Возвращаемое значение

true в случае успеха, false - если нет возможности изменить элемент.

### Пример:

```
//--- example for CArrayShort::Update(int,short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- update element
    if(!array.Update(0,100))
    {
        printf("Update error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shift

Перемещает элемент из указанной позиции массива на указанное смещение.

```
bool Shift(
    int pos,           // позиция
    int shift          // смещение
)
```

### Параметры

*pos*

[in] Позиция перемещаемого элемента массиве

*shift*

[in] Значение смещения (как положительное, так и отрицательное значение).

### Возвращаемое значение

true в случае успеха, false - если нет возможности переместить элемент.

### Пример:

```
//--- example for CArrayShort::Shift(int,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shift element
    if(!array.Shift(10,-5))
    {
        printf("Shift error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Delete

Удаляет элемент из указанной позиции массива.

```
bool Delete(  
    int pos // позиция  
)
```

### Параметры

*pos*

[in] Позиция удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элемент.

### Пример:

```
//--- example for CArrayShort::Delete(int)  
#include <Arrays\ArrayShort.mqh>  
//---  
void OnStart()  
{  
    CArrayShort *array=new CArrayShort;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```

## DeleteRange

Удаляет группу элементов из указанной позиции массива.

```
bool DeleteRange(
    int   from,      // позиция первого элемента
    int   to        // позиция последнего элемента
)
```

### Параметры

*from*

[in] Позиция первого удаляемого элемента в массиве.

*to*

[in] Позиция последнего удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элементы.

### Пример:

```
//--- example for CArrayShort::DeleteRange(int,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- delete elements
    if(!array.DeleteRange(0,10))
    {
        printf("Delete error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## At

Получает элемент из указанной позиции массива.

```
short At(
    int pos        // позиция
) const
```

### Параметры

*pos*

[in] Позиция искомого элемента в массиве.

### Возвращаемое значение

Значение элемента в случае успеха, `SHORT_MAX`- если была попытка получить элемент из не существующей позиции (при этом код последней ошибки `ERR_OUT_OF_RANGE`).

### Примечание

Разумеется, `SHORT_MAX` может быть и валидным значением элемента массива, поэтому, получив такое значение, всегда проверяйте код последней ошибки.

### Пример:

```
//--- example for CArrayShort::At(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    for(int i=0;i<array.Total();i++)
    {
        short result=array.At(i);
        if(result==SHORT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
            //--- ошибка чтения из массива
            printf("Get element error");
            delete array;
            return;
        }
    }
    //--- use element
```

```
//--- . . .
}
//--- delete array
delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const short& src[]          // массив-источник
) const
```

### Параметры

*src []*  
[in] Ссылка на массив-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayShort::CompareArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete array
    delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const CArrayShort* src          // указатель на источник
) const
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayShort-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayShort::CompareArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayShort *src=new CArrayShort;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete arrays
    delete src;
    delete array;
}
```

## InsertSort

Вставляет элемент в сортированный массив.

```
bool InsertSort(
    short element      // элемент для вставки
)
```

### Параметры

*element*

[in] Значение элемента для вставки в сортированный массив

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayShort::InsertSort(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- insert element
    if(!array.InsertSort(100))
    {
        printf("Insert error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Search

Ищет элемент равный образцу в сортированном массиве.

```
int Search(
    short element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayShort::Search(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.Search(100)!=-1) printf("Element found");
    else                      printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreat

Ищет элемент больше образца в сортированном массиве.

```
int SearchGreat(
    short element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayShort::SearchGreat(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreat(100)!=-1) printf("Element found");
    else                           printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLess

Ищет элемент меньше образца в сортированном массиве.

```
int SearchLess(
    short element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayShort::SearchLess(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLess(100)!=-1) printf("Element found");
    else                           printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreatOrEqual

Ищет элемент больше или равный образцу в сортированном массиве.

```
int SearchGreatOrEqual(
    short element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayShort::SearchGreatOrEqual(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreatOrEqual(100)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLessOrEqual

Ищет элемент меньше или равный образцу в сортированном массиве.

```
int SearchLessOrEqual(
    short element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayShort::SearchLessOrEqual(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLessOrEqual(100)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchFirst

Ищет первый элемент равный образцу в сортированном массиве.

```
int SearchFirst(
    short element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayShort::SearchFirst(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchFirst(100)!=-1) printf("Element found");
    else                           printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLast

Ищет последний элемент равный образцу в сортированном массиве.

```
int SearchLast(
    short element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayShort::SearchLast(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLast(100)!=-1) printf("Element found");
    else                         printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLinear

Ищет элемент, равный образцу в массиве.

```
int SearchLinear(
    short element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Примечание

Поиск производится последовательным перебором элементов массива (линейный поиск для несортированных массивов).

### Пример:

```
//--- example for CArrayShort::SearchLinear(short)
#include <Arrays\ArrayShort.mqh>
//---

void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- search element
    if(array.SearchLinear(100)!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## Save

Сохраняет данные массива в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayShort::Save(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    int          file_handle;
    CArrayShort *array=new CArrayShort;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add 100 arrays elements
    for(int i=0;i<100;i++)
    {
        array.Add(i);
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
        }
    }
}
```

```
    }
    FileClose(file_handle);
}
delete array;
}
```

## Load

Загружает данные массива из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayShort::Load(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    int          file_handle;
    CArrayShort *array=new CArrayShort;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- use arrays elements
    for(int i=0;i<array.Total();i++)
```

```
{  
    printf("Element[%d] = %d", i, array.At(i));  
}  
delete array;  
}
```

## Type

Получает идентификатор типа массива.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа массива (для CArrayShort - 82).

### Пример:

```
//--- example for CArrayShort::Type()
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
{
    CArrayShort *array=new CArrayShort;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```

## CArrayInt

Класс CArrayInt является классом динамического массива переменных типа int или uint.

### Описание

Класс CArrayInt обеспечивает возможность работы с динамическим массивом переменных типа int или uint. В классе реализованы возможности добавления/вставки/удаления элементов массива, сортировки массива, поисков в отсортированном массиве. Кроме того, реализованы методы работы с файлом.

### Декларация

```
class CArrayInt : public CArray
```

### Заголовок

```
#include <Arrays\ArrayInt.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayInt
```

### Прямые потомки

[CSpreadBuffer](#)

### Методы класса по группам

Управление памятью	
<a href="#">Reserve</a>	Резервирует память для увеличения размеров массива
<a href="#">Resize</a>	Устанавливает новый (меньший) размер массива
<a href="#">Shutdown</a>	Очищает массив с полным освобождением памяти
Наполнение	
<a href="#">Add</a>	Добавляет элемент в конец массива
<a href="#">AddArray</a>	Добавляет в конец массива элементы из другого массива
<a href="#">AddArray</a>	Добавляет в конец массива элементы из другого массива
<a href="#">Insert</a>	Вставляет элемент в массив в указанную позицию

<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<b>Изменение</b>	
<a href="#"><u>Update</u></a>	Изменяет элемент в указанной позиции массива
<a href="#"><u>Shift</u></a>	Перемещает элемент из указанной позиции массива на указанное смещение
<b>Удаление</b>	
<a href="#"><u>Delete</u></a>	Удаляет элемент из указанной позиции массива
<a href="#"><u>DeleteRange</u></a>	Удаляет группу элементов из указанной позиции массива
<b>Доступ</b>	
<a href="#"><u>At</u></a>	Получает элемент из указанной позиции массива
<b>Сравнение</b>	
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<b>Операции с отсортированным массивом</b>	
<a href="#"><u>InsertSort</u></a>	Вставляет элемент в отсортированный массив
<a href="#"><u>Search</u></a>	Ищет элемент равный образцу в отсортированном массиве
<a href="#"><u>SearchGreat</u></a>	Ищет элемент больше образца в отсортированном массиве
<a href="#"><u>SearchLess</u></a>	Ищет элемент меньше образца в отсортированном массиве
<a href="#"><u>SearchGreatOrEqual</u></a>	Ищет элемент больше или равный образцу в отсортированном массиве
<a href="#"><u>SearchLessOrEqual</u></a>	Ищет элемент меньше или равный образцу в отсортированном массиве

<a href="#"><u>SearchFirst</u></a>	Ищет первый элемент равный образцу в сортированном массиве
<a href="#"><u>SearchLast</u></a>	Ищет последний элемент равный образцу в сортированном массиве
<a href="#"><u>SearchLinear</u></a>	Ищет элемент равный образцу в массиве
<b>Ввод/вывод</b>	
<b>virtual <a href="#"><u>Save</u></a></b>	Сохраняет данные массива в файле
<b>virtual <a href="#"><u>Load</u></a></b>	Загружает данные массива из файла
<b>virtual <a href="#"><u>Type</u></a></b>	Получает идентификатор типа массива

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

**Методы унаследованные от CArray**

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

## Reserve

Резервирует память для увеличения размеров массива.

```
bool Reserve(
    int size        // количество
)
```

### Параметры

*size*

[in] Количество дополнительных элементов массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка запросить количество меньше или равное нулю, либо если массив увеличить не удалось.

### Примечание

Для уменьшения фрагментации памяти, приращение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayInt::Reserve(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## Resize

Устанавливает новый (меньший) размер массива.

```
bool Resize(
    int size        // размер
)
```

### Параметры

*size*

[in] Новый размер массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка установить размер меньше нуля.

### Примечание

Изменение размера массива позволяет оптимально использовать память. Лишние элементы справа теряются. Для уменьшения фрагментации памяти, изменение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayInt::Resize(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- resize array
    if(!array.Resize(10))
    {
        printf("Resize error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shutdown

Очищает массив с полным освобождением памяти.

```
bool Shutdown()
```

### Возвращаемое значение

true в случае успеха, false - если произошла ошибка.

### Пример:

```
//--- example for CArrayInt::Shutdown()
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Add

Добавляет элемент в конец массива.

```
bool Add(
    int element      // элемент для добавления
)
```

### Параметры

*element*

[in] Значение элемента для добавления в массив.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элемент.

### Пример:

```
//--- example for CArrayInt::Add(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    for(int i=0;i<100;i++)
    {
        if(!array.Add(i))
        {
            printf("Element addition error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const int& src[]          // массив-источник
)
```

### Параметры

*src []*  
 [in] Ссылка на массив-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayInt::AddArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---

void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const CArrayInt* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayInt-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayInt::AddArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayInt *src=new CArrayInt;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Insert

Вставляет элемент в массив в указанную позицию.

```
bool Insert(
    int element,      // элемент для вставки
    int pos          // позиция
)
```

### Параметры

*element*

[in] Значение элемента для вставки в массив

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayInt::Insert(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert elements
    for(int i=0;i<100;i++)
    {
        if(!array.Insert(i,0))
        {
            printf("Insert error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    const int& src[],      // массив-источник
    int          pos        // позиция
)
```

### Параметры

*src[]*  
 [in] Ссылка на массив-источник элементов для вставки

*pos*  
 [in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayInt::InsertArray(const int &[],int)
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    CArrayInt* src,          // указатель на источник
    int      pos             // позиция
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayInt-источник элементов для вставки.

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayInt::InsertArray(const CArrayInt*,int)
#include <Arrays\ArrayInt.mqh>
//---

void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayInt *src=new CArrayInt;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete src;
    }
}
```

```
    delete array;
    return;
}
//--- delete source array
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const int& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayInt::AssignArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---

void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const CArrayInt* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayInt-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayInt::AssignArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---

void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---

    if(array==NULL)
    {
        printf("Object create error");
        return;
    }

    //--- create source array
    CArrayInt *src =new CArrayInt;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }

    //--- add source arrays elements
    //--- . . .
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- arrays is identical  
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Update

Изменяет элемент в указанной позиции массива.

```
bool Update(
    int pos,           // позиция
    int element        // значение
)
```

### Параметры

*pos*  
 [in] Позиция элемента в массиве для изменения

*element*  
 [in] Новое значение элемента

### Возвращаемое значение

true в случае успеха, false - если нет возможности изменить элемент.

### Пример:

```
//--- example for CArrayInt::Update(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- update element
    if(!array.Update(0,10000))
    {
        printf("Update error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shift

Перемещает элемент из указанной позиции массива на указанное смещение.

```
bool Shift(
    int pos,           // позиция
    int shift          // смещение
)
```

### Параметры

*pos*

[in] Позиция перемещаемого элемента массиве

*shift*

[in] Значение смещения (как положительное, так и отрицательное значение).

### Возвращаемое значение

true в случае успеха, false - если нет возможности переместить элемент.

### Пример:

```
//--- example for CArrayInt::Shift(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shift element
    if(!array.Shift(10,-5))
    {
        printf("Shift error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Delete

Удаляет элемент из указанной позиции массива.

```
bool Delete(  
    int pos        // позиция  
)
```

### Параметры

*pos*

[in] Позиция удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элемент.

### Пример:

```
//--- example for CArrayInt::Delete(int)  
#include <Arrays\ArrayInt.mqh>  
//---  
void OnStart()  
{  
    CArrayInt *array=new CArrayInt;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```

## DeleteRange

Удаляет группу элементов из указанной позиции массива.

```
bool DeleteRange(
    int from,          // позиция первого элемента
    int to            // позиция последнего элемента
)
```

### Параметры

*from*

[in] Позиция первого удаляемого элемента в массиве.

*to*

[in] Позиция последнего удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элементы.

### Пример:

```
//--- example for CArrayInt::DeleteRange(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- delete elements
    if(!array.DeleteRange(0,10))
    {
        printf("Delete error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## At

Получает элемент из указанной позиции массива.

```
int At(
    int pos      // позиция
) const
```

### Параметры

*pos*

[in] Позиция искомого элемента в массиве.

### Возвращаемое значение

Значение элемента в случае успеха, INT\_MAX- если была попытка получить элемент из не существующей позиции (при этом код последней ошибки ERR\_OUT\_OF\_RANGE).

### Примечание

Разумеется, INT\_MAX может быть и валидным значением элемента массива, поэтому, получив такое значение, всегда проверяйте код последней ошибки.

### Пример:

```
//--- example for CArrayInt::At(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    for(int i=0;i<array.Total();i++)
    {
        int result=array.At(i);
        if(result==INT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
            //--- ошибка чтения из массива
            printf("Get element error");
            delete array;
            return;
        }
    }
    //--- use element
```

```
//--- . . .
}
//--- delete array
delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const int& src[]          // массив-источник
) const
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayInt::CompareArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete array
    delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const CArrayInt* src          // указатель на источник
) const
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayInt-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayInt::CompareArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayInt *src=new CArrayInt;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete arrays
    delete src;
    delete array;
}
```

## InsertSort

Вставляет элемент в сортированный массив.

```
bool InsertSort(
    int element           // элемент для вставки
)
```

### Параметры

*element*

[in] Значение элемента для вставки в сортированный массив

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayInt::InsertSort(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- insert element
    if(!array.InsertSort(10000))
    {
        printf("Insert error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Search

Ищет элемент равный образцу в сортированном массиве.

```
int Search(
    int element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayInt::Search(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.Search(10000)!=-1) printf("Element found");
    else                         printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreat

Ищет элемент больше образца в сортированном массиве.

```
int SearchGreat(
    int element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayInt::SearchGreat(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreat(10000)!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLess

Ищет элемент меньше образца в сортированном массиве.

```
int SearchLess(
    int element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayInt::SearchLess(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLess(10000)!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreatOrEqual

Ищет элемент больше или равный образцу в сортированном массиве.

```
int SearchGreatOrEqual(
    int element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayInt::SearchGreatOrEqual(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreatOrEqual(10000)!=-1) printf("Element found");
    else                                     printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLessOrEqual

Ищет элемент меньше или равный образцу в сортированном массиве.

```
int SearchLessOrEqual(
    int element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayInt::SearchLessOrEqual(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLessOrEqual(10000)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchFirst

Ищет первый элемент равный образцу в сортированном массиве.

```
int SearchFirst(
    int element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayInt:: SearchFirst(int)
#include <Arrays\ArrayInt.mqh>
//---

void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchFirst(10000)!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLast

Ищет последний элемент равный образцу в сортированном массиве.

```
int SearchLast(
    int element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayInt::SearchLast(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLast(10000)!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLinear

Ищет элемент равный образцу в массиве.

```
int SearchLinear(
    int element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Примечание

Поиск производится последовательным перебором элементов массива (линейный поиск для несортированных массивов).

### Пример:

```
//--- example for CArrayInt::SearchLinear(int)
#include <Arrays\ArrayInt.mqh>
//---

void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- search element
    if(array.SearchLinear(10000)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## Save

Сохраняет данные массива в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayInt::Save(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    int         file_handle;
    CArrayInt *array=new CArrayInt;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add 100 arrays elements
    for(int i=0;i<100;i++)
    {
        array.Add(i);
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
        }
    }
}
```

```
    }
    FileClose(file_handle);
}
delete array;
}
```

## Load

Загружает данные массива из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayInt::Load(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    int      file_handle;
    CArrayInt *array=new CArrayInt;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- use arrays elements
    for(int i=0;i<array.Total();i++)
```

```
{  
    printf("Element[%d] = %d", i, array.At(i));  
}  
delete array;  
}
```

## Type

Получает идентификатор типа массива.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа массива (для CArrayInt - 82).

### Пример:

```
//--- example for CArrayInt::Type()
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
{
    CArrayInt *array=new CArrayInt;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```

## CArrayLong

Класс CArrayLong является классом динамического массива переменных типа long или ulong.

### Описание

Класс CArrayLong обеспечивает возможность работы с динамическим массивом переменных типа long или ulong. В классе реализованы возможности добавления/вставки/удаления элементов массива, сортировки массива, поисков в отсортированном массиве. Кроме того, реализованы методы работы с файлом.

### Декларация

```
class CArrayLong : public CArray
```

### Заголовок

```
#include <Arrays\ArrayLong.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayLong
```

### Прямые потомки

[CRealVolumeBuffer](#), [CTickVolumeBuffer](#), [CTimeBuffer](#)

### Методы класса по группам

Управление памятью	
<a href="#">Reserve</a>	Резервирует память для увеличения размеров массива
<a href="#">Resize</a>	Устанавливает новый (меньший) размер массива
<a href="#">Shutdown</a>	Очищает массив с полным освобождением памяти
Наполнение	
<a href="#">Add</a>	Добавляет элемент в конец массива
<a href="#">AddArray</a>	Добавляет в конец массива элементы из другого массива
<a href="#">AddArray</a>	Добавляет в конец массива элементы из другого массива
<a href="#">Insert</a>	Вставляет элемент в массив в указанную позицию

<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<b>Изменение</b>	
<a href="#"><u>Update</u></a>	Изменяет элемент в указанной позиции массива
<a href="#"><u>Shift</u></a>	Перемещает элемент из указанной позиции массива на указанное смещение
<b>Удаление</b>	
<a href="#"><u>Delete</u></a>	Удаляет элемент из указанной позиции массива
<a href="#"><u>DeleteRange</u></a>	Удаляет группу элементов из указанной позиции массива
<b>Доступ</b>	
<a href="#"><u>At</u></a>	Получает элемент из указанной позиции массива
<b>Сравнение</b>	
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<b>Операции с отсортированным массивом</b>	
<a href="#"><u>InsertSort</u></a>	Вставляет элемент в отсортированный массив
<a href="#"><u>Search</u></a>	Ищет элемент равный образцу в отсортированном массиве
<a href="#"><u>SearchGreat</u></a>	Ищет элемент больше образца в отсортированном массиве
<a href="#"><u>SearchLess</u></a>	Ищет элемент меньше образца в отсортированном массиве
<a href="#"><u>SearchGreatOrEqual</u></a>	Ищет элемент больше или равный образцу в отсортированном массиве
<a href="#"><u>SearchLessOrEqual</u></a>	Ищет элемент меньше или равный образцу в отсортированном массиве

<a href="#"><u>SearchFirst</u></a>	Ищет первый элемент равный образцу в сортированном массиве
<a href="#"><u>SearchLast</u></a>	Ищет последний элемент равный образцу в сортированном массиве
<a href="#"><u>SearchLinear</u></a>	Ищет элемент равный образцу в массиве
<b>Ввод/вывод</b>	
<b>virtual <a href="#"><u>Save</u></a></b>	Сохраняет данные массива в файле
<b>virtual <a href="#"><u>Load</u></a></b>	Загружает данные массива из файла
<b>virtual <a href="#"><u>Type</u></a></b>	Получает идентификатор типа массива

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

**Методы унаследованные от CArray**

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

## Reserve

Резервирует память для увеличения размеров массива.

```
bool Reserve(
    int size        // количество
)
```

### Параметры

*size*

[in] Количество дополнительных элементов массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка запросить количество меньше или равное нулю, либо если массив увеличить не удалось.

### Примечание

Для уменьшения фрагментации памяти, приращение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayLong::Reserve(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## Resize

Устанавливает новый (меньший) размер массива.

```
bool Resize(
    int size        // размер
)
```

### Параметры

*size*

[in] Новый размер массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка установить размер меньше нуля.

### Примечание

Изменение размера массива позволяет оптимально использовать память. Лишние элементы справа теряются. Для уменьшения фрагментации памяти, изменение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayLong::Resize(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- resize array
    if(!array.Resize(10))
    {
        printf("Resize error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shutdown

Очищает массив с полным освобождением памяти.

```
bool Shutdown()
```

### Возвращаемое значение

true в случае успеха, false - если произошла ошибка.

### Пример:

```
//--- example for CArrayLong::Shutdown()
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Add

Добавляет элемент в конец массива.

```
bool Add(
    long element      // элемент для добавления
)
```

### Параметры

*element*

[in] Значение элемента для добавления в массив.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элемент.

### Пример:

```
//--- example for CArrayLong::Add(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    for(int i=0;i<100;i++)
    {
        if(!array.Add(i))
        {
            printf("Element addition error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const long& src[]          // массив-источник
)
```

### Параметры

*src []*  
 [in] Ссылка на массив-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayLong::AddArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const CArrayLong* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayLong-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayLong::AddArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---

void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---

    if(array==NULL)
    {
        printf("Object create error");
        return;
    }

    //--- create source array
    CArrayLong *src=new CArrayLong;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }

    //--- add source arrays elements
    //--- . . .
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Insert

Вставляет элемент в массив в указанную позицию.

```
bool Insert(
    long element,      // элемент для вставки
    int pos           // позиция
)
```

### Параметры

*element*

[in] Значение элемента для вставки в массив

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayLong::Insert(long,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert elements
    for(int i=0;i<100;i++)
    {
        if(!array.Insert(i,0))
        {
            printf("Insert error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    const long& src[],      // массив-источник
    int          pos        // позиция
)
```

### Параметры

*src[]*  
 [in] Ссылка на массив-источник элементов для вставки

*pos*  
 [in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayLong::InsertArray(const long &[],int)
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    CArrayLong* src,          // указатель на источник
    int pos                  // позиция
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayLong-источник элементов для вставки.

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayLong::InsertArray(const CArrayLong*,int)
#include <Arrays\ArrayLong.mqh>
//---

void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayLong *src=new CArrayLong;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete src;
    }
}
```

```
    delete array;
    return;
}
//--- delete source array
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const long& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayLong::AssignArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---

void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const CArrayLong* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayLong-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayLong::AssignArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---

void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayLong *src =new CArrayLong;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- arrays is identical  
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Update

Изменяет элемент в указанной позиции массива.

```
bool Update(
    int pos,           // позиция
    long element       // значение
)
```

### Параметры

*pos*  
 [in] Позиция элемента в массиве для изменения

*element*  
 [in] Новое значение элемента

### Возвращаемое значение

true в случае успеха, false - если нет возможности изменить элемент.

### Пример:

```
//--- example for CArrayLong::Update(int,long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- update element
    if(!array.Update(0,1000000))
    {
        printf("Update error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shift

Перемещает элемент из указанной позиции массива на указанное смещение.

```
bool Shift(
    int pos,           // позиция
    int shift          // смещение
)
```

### Параметры

*pos*

[in] Позиция перемещаемого элемента массиве

*shift*

[in] Значение смещения (как положительное, так и отрицательное значение).

### Возвращаемое значение

true в случае успеха, false - если нет возможности переместить элемент.

### Пример:

```
//--- example for CArrayLong::Shift(int,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shift element
    if(!array.Shift(10,-5))
    {
        printf("Shift error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Delete

Удаляет элемент из указанной позиции массива.

```
bool Delete(
    int pos        // позиция
)
```

### Параметры

*pos*

[in] Позиция удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элемент.

### Пример:

```
//--- example for CArrayLong::Delete(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- delete element
    if(!array.Delete(0))
    {
        printf("Delete error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## DeleteRange

Удаляет группу элементов из указанной позиции массива.

```
bool DeleteRange(
    int   from,      // позиция первого элемента
    int   to        // позиция последнего элемента
)
```

### Параметры

*from*

[in] Позиция первого удаляемого элемента в массиве.

*to*

[in] Позиция последнего удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элементы.

### Пример:

```
//--- example for CArrayLong::DeleteRange(int,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- delete elements
    if(!array.DeleteRange(0,10))
    {
        printf("Delete error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## At

Получает элемент из указанной позиции массива.

```
long At(
    int pos        // позиция
) const
```

### Параметры

*pos*

[in] Позиция искомого элемента в массиве.

### Возвращаемое значение

Значение элемента в случае успеха, LONG\_MAX если была попытка получить элемент из не существующей позиции (при этом код последней ошибки ERR\_OUT\_OF\_RANGE).

### Примечание

Разумеется, LONG\_MAX может быть и валидным значением элемента массива, поэтому, получив такое значение, всегда проверяйте код последней ошибки.

### Пример:

```
//--- example for CArrayLong::At(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    for(int i=0;i<array.Total();i++)
    {
        long result=array.At(i);
        if(result==LONG_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
            //--- ошибка чтения из массива
            printf("Get element error");
            delete array;
            return;
        }
    }
    //--- use element
```

```
//--- . . .
}
//--- delete array
delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const long& src[]          // массив-источник
) const
```

### Параметры

*src []*  
 [in] Ссылка на массив-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayLong::CompareArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete array
    delete array;
}
```

## CompareArrayconst

Сравнивает массив с другим массивом.

```
bool CompareArrayconst(
    const CArrayLong* src          // указатель на источник
) const
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayLong-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayLong::CompareArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayLong *src=new CArrayLong;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete arrays
    delete src;
    delete array;
}
```

## InsertSort

Вставляет элемент в сортированный массив.

```
bool InsertSort(
    long element      // элемент для вставки
)
```

### Параметры

*element*

[in] Значение элемента для вставки в сортированный массив

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayLong::InsertSort(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- insert element
    if(!array.InsertSort(1000000))
    {
        printf("Insert error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Search

Ищет элемент равный образцу в сортированном массиве.

```
int Search(
    long element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayLong::Search(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.Search(1000000)!=-1) printf("Element found");
    else                           printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreat

Ищет элемент больше образца в сортированном массиве.

```
int SearchGreat(
    long element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayLong::SearchGreat(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreat(1000000)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLess

Ищет элемент меньше образца в сортированном массиве.

```
int SearchLess(
    long element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayLong::SearchLess(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLess(1000000)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreatOrEqual

Ищет элемент больше или равный образцу в сортированном массиве.

```
int SearchGreatOrEqual(
    long element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayLong::SearchGreatOrEqual(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreatOrEqual(1000000)!=-1) printf("Element found");
    else                                         printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLessOrEqual

Ищет элемент меньше или равный образцу в сортированном массиве.

```
int SearchLessOrEqual(
    long element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayLong::SearchLessOrEqual(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLessOrEqual(1000000)!=-1) printf("Element found");
    else                                     printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchFirst

Ищет первый элемент равный образцу в сортированном массиве.

```
int SearchFirst(
    long element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayLong::SearchFirst(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchFirst(1000000)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLast

Ищет последний элемент равный образцу в сортированном массиве.

```
int SearchLast(
    long element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayLong::SearchLast(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLast(1000000)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLinear

Ищет элемент равный образцу в массиве.

```
int SearchLinear(
    long element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Примечание

Поиск производится последовательным перебором элементов массива (линейный поиск для несорттированных массивов).

### Пример:

```
//--- example for CArrayLong::SearchLinear(long)
#include <Arrays\ArrayLong.mqh>
//---

void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- search element
    if(array.SearchLinear(1000000)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## Save

Сохраняет данные массива в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayLong::Save(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    int         file_handle;
    CArrayLong *array=new CArrayLong;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add 100 arrays elements
    for(int i=0;i<100;i++)
    {
        array.Add(i);
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
        }
    }
}
```

```
    }
    FileClose(file_handle);
}
delete array;
}
```

## Load

Загружает данные массива из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayLong::Load(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    int         file_handle;
    CArrayLong *array=new CArrayLong;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- use arrays elements
    for(int i=0;i<array.Total();i++)
```

```
{  
    printf("Element[%d] = %I64", i, array.At(i));  
}  
delete array;  
}
```

## Type

Получает идентификатор типа массива.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа массива (для CArrayLong - 84).

### Пример:

```
//--- example for CArrayLong::Type()
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
{
    CArrayLong *array=new CArrayLong;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```

## CArrayFloat

Класс CArrayFloat является классом динамического массива переменных типа float.

### Описание

Класс CArrayFloat обеспечивает возможность работы с динамическим массивом переменных типа float. В классе реализованы возможности добавления/вставки/удаления элементов массива, сортировки массива, поисков в отсортированном массиве. Кроме того, реализованы методы работы с файлом.

### Декларация

```
class CArrayFloat : public CArray
```

### Заголовок

```
#include <Arrays\ArrayFloat.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayFloat
```

### Методы класса по группам

Атрибуты	
<a href="#"><u>Delta</u></a>	Устанавливает величину допуска сравнения
<a href="#"><u>Управление памятью</u></a>	
<a href="#"><u>Reserve</u></a>	Резервирует память для увеличения размеров массива
<a href="#"><u>Resize</u></a>	Устанавливает новый (меньший) размер массива
<a href="#"><u>Shutdown</u></a>	Очищает массив с полным освобождением памяти
<a href="#"><u>Наполнение</u></a>	
<a href="#"><u>Add</u></a>	Добавляет элемент в конец массива
<a href="#"><u>AddArray</u></a>	Добавляет в конец массива элементы из другого массива
<a href="#"><u>AddArray</u></a>	Добавляет в конец массива элементы из другого массива
<a href="#"><u>Insert</u></a>	Вставляет элемент в массив в указанную позицию

<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<b>Изменение</b>	
<a href="#"><u>Update</u></a>	Изменяет элемент в указанной позиции массива
<a href="#"><u>Shift</u></a>	Перемещает элемент из указанной позиции массива на указанное смещение
<b>Удаление</b>	
<a href="#"><u>Delete</u></a>	Удаляет элемент из указанной позиции массива
<a href="#"><u>DeleteRange</u></a>	Удаляет группу элементов из указанной позиции массива
<b>Доступ</b>	
<a href="#"><u>At</u></a>	Получает элемент из указанной позиции массива
<b>Сравнение</b>	
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<b>Операции с отсортированным массивом</b>	
<a href="#"><u>InsertSort</u></a>	Вставляет элемент в отсортированный массив
<a href="#"><u>Search</u></a>	Ищет элемент равный образцу в отсортированном массиве
<a href="#"><u>SearchGreat</u></a>	Ищет элемент больше образца в отсортированном массиве
<a href="#"><u>SearchLess</u></a>	Ищет элемент меньше образца в отсортированном массиве
<a href="#"><u>SearchGreatOrEqual</u></a>	Ищет элемент больше или равный образцу в отсортированном массиве
<a href="#"><u>SearchLessOrEqual</u></a>	Ищет элемент меньше или равный образцу в отсортированном массиве

<a href="#"><u>SearchFirst</u></a>	Ищет первый элемент равный образцу в сортированном массиве
<a href="#"><u>SearchLast</u></a>	Ищет последний элемент равный образцу в сортированном массиве
<a href="#"><u>SearchLinear</u></a>	Ищет элемент равный образцу в массиве
<b>Ввод/вывод</b>	
<b>virtual <a href="#"><u>Save</u></a></b>	Сохраняет данные массива в файле
<b>virtual <a href="#"><u>Load</u></a></b>	Загружает данные массива из файла
<b>virtual <a href="#"><u>Type</u></a></b>	Получает идентификатор типа массива

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

**Методы унаследованные от CArray**

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

## Delta

Устанавливает допуск сравнения.

```
void Delta(
    float delta      // допуск
)
```

### Параметры

*delta*  
[in] новое значение допуска сравнения.

### Возвращаемое значение

Нет

### Примечание

Допуск сравнения используется при поисках. Значения считаются равными, если их разница меньше или равна допуску. По умолчанию допуск равен 0.0.

### Пример:

```
//--- example for CArrayFloat::Delta(float)
#include <Arrays\ArrayFloat.mqh>
//---

void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---

    if(array==NULL)
    {
        printf("Object create error");
        return;
    }

    //--- set compare variation
    array.Delta(0.001);
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## Reserve

Резервирует память для увеличения размеров массива.

```
bool Reserve(
    int size        // количество
)
```

### Параметры

*size*

[in] Количество дополнительных элементов массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка запросить количество меньше или равное нулю, либо если массив увеличить не удалось.

### Примечание

Для уменьшения фрагментации памяти, приращение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayFloat::Reserve(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## Resize

Устанавливает новый (меньший) размер массива.

```
bool Resize(
    int size        // размер
)
```

### Параметры

*size*

[in] Новый размер массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка установить размер меньше нуля.

### Примечание

Изменение размера массива позволяет оптимально использовать память. Лишние элементы справа теряются. Для уменьшения фрагментации памяти, изменение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayFloat::Resize(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- resize array
    if(!array.Resize(10))
    {
        printf("Resize error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shutdown

Очищает массив с полным освобождением памяти.

```
bool Shutdown()
```

### Возвращаемое значение

true в случае успеха, false - если произошла ошибка.

### Пример:

```
//--- example for CArrayFloat::Shutdown()
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Add

Добавляет элемент в конец массива.

```
bool Add(
    float element      // элемент для добавления
)
```

### Параметры

*element*

[in] Значение элемента для добавления в массив.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элемент.

### Пример:

```
//--- example for CArrayFloat::Add(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    for(int i=0;i<100;i++)
    {
        if(!array.Add(i))
        {
            printf("Element addition error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const float& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayFloat::AddArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const CArrayFloat* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayFloat-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayFloat::AddArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---

void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayFloat *src=new CArrayFloat;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Insert

Вставляет элемент в массив в указанную позицию.

```
bool Insert(
    float element,      // элемент для вставки
    int pos            // позиция
)
```

### Параметры

*element*

[in] Значение элемента для вставки в массив

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayFloat::Insert(float,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert elements
    for(int i=0;i<100;i++)
    {
        if(!array.Insert(i,0))
        {
            printf("Insert error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    const float& src[],      // массив-источник
    int           pos         // позиция
)
```

### Параметры

*src[]*  
 [in] Ссылка на массив-источник элементов для вставки

*pos*  
 [in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayFloat::InsertArray(const float &[],int)
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    CArrayFloat* src,          // указатель на источник
    int pos                   // позиция
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayFloat-источник элементов для вставки.

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayFloat::InsertArray(const CArrayFloat*,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayFloat *src=new CArrayFloat;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete src;
    }
}
```

```
    delete array;
    return;
}
//--- delete source array
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const float& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayFloat::AssignArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const CArrayFloat* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayFloat-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayFloat::AssignArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---

void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayFloat *src =new CArrayFloat;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- arrays is identical  
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Update

Изменяет элемент в указанной позиции массива.

```
bool Update(
    int pos,           // позиция
    float element      // значение
)
```

### Параметры

*pos*  
 [in] Позиция элемента в массиве для изменения

*element*  
 [in] Новое значение элемента

### Возвращаемое значение

true в случае успеха, false - если нет возможности изменить элемент.

### Пример:

```
//--- example for CArrayFloat::Update(int,float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- update element
    if(!array.Update(0,100.0))
    {
        printf("Update error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shift

Перемещает элемент из указанной позиции массива на указанное смещение.

```
bool Shift(
    int pos,           // позиция
    int shift          // смещение
)
```

### Параметры

*pos*

[in] Позиция перемещаемого элемента массиве

*shift*

[in] Значение смещения (как положительное, так и отрицательное значение).

### Возвращаемое значение

true в случае успеха, false - если нет возможности переместить элемент.

### Пример:

```
//--- example for CArrayFloat::Shift(int,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shift element
    if(!array.Shift(10,-5))
    {
        printf("Shift error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Delete

Удаляет элемент из указанной позиции массива.

```
bool Delete(  
    int pos // позиция  
)
```

### Параметры

*pos*

[in] Позиция удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элемент.

### Пример:

```
//--- example for CArrayFloat::Delete(int)  
#include <Arrays\ArrayFloat.mqh>  
//---  
void OnStart()  
{  
    CArrayFloat *array=new CArrayFloat;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```

## DeleteRange

Удаляет группу элементов из указанной позиции массива.

```
bool DeleteRange(
    int   from,      // позиция первого элемента
    int   to        // позиция последнего элемента
)
```

### Параметры

*from*

[in] Позиция первого удаляемого элемента в массиве.

*to*

[in] Позиция последнего удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элементы.

### Пример:

```
//--- example for CArrayFloat::DeleteRange(int,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- delete elements
    if(!array.DeleteRange(0,10))
    {
        printf("Delete error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## At

Получает элемент из указанной позиции массива.

```
float At(
    int pos        // позиция
) const
```

### Параметры

*pos*

[in] Позиция искомого элемента в массиве.

### Возвращаемое значение

Значение элемента в случае успеха, `FLT_MAX` если была попытка получить элемент из не существующей позиции (при этом код последней ошибки `ERR_OUT_OF_RANGE`).

### Примечание

Разумеется, `FLT_MAX` может быть и валидным значением элемента массива, поэтому, получив такое значение, всегда проверяйте код последней ошибки.

### Пример:

```
//--- example for CArrayFloat::At(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    for(int i=0;i<array.Total();i++)
    {
        float result=array.At(i);
        if(result==FLT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
            //--- ошибка чтения из массива
            printf("Get element error");
            delete array;
            return;
        }
    }
    //--- use element
```

```
//--- . . .
}
//--- delete array
delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const float& src[]          // массив-источник
) const
```

### Параметры

*src []*  
[in] Ссылка на массив-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayFloat::CompareArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete array
    delete array;
}
```

## AssignArrayconst

Копирует в массив элементы из другого массива.

```
bool AssignArrayconst(
    const CArrayFloat* src          // указатель на источник
) const
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayFloat-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayFloat::CompareArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---

void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayFloat *src=new CArrayFloat;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete arrays
    delete src;
    delete array;
}
```

## InsertSort

Вставляет элемент в сортированный массив.

```
bool InsertSort(
    float element      // элемент для вставки
)
```

### Параметры

*element*

[in] Значение элемента для вставки в сортированный массив

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayFloat::InsertSort(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- insert element
    if(!array.InsertSort(100.0))
    {
        printf("Insert error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Search

Ищет элемент равный образцу в сортированном массиве.

```
int Search(
    float element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayFloat::Search(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.Search(100.0)!=-1) printf("Element found");
    else                         printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreat

Ищет элемент больше образца в сортированном массиве.

```
int SearchGreat(
    float element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayFloat::SearchGreat(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreat(100.0)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLess

Ищет элемент меньше образца в сортированном массиве.

```
int SearchLess(
    float element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayFloat:: SearchLess(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLess(100.0)!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreatOrEqual

Ищет элемент больше или равный образцу в сортированном массиве.

```
int SearchGreatOrEqual(
    float element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayFloat::SearchGreatOrEqual(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreatOrEqual(100.0)!=-1) printf("Element found");
    else                                     printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLessOrEqual

Ищет элемент меньше или равный образцу в сортированном массиве.

```
int SearchLessOrEqual(
    float element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayFloat::SearchLessOrEqual(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLessOrEqual(100.0)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchFirst

Ищет первый элемент равный образцу в сортированном массиве.

```
int SearchFirst(
    float element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayFloat::SearchFirst(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchFirst(100.0)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLast

Ищет последний элемент равный образцу в сортированном массиве.

```
int SearchLast(
    float element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayFloat::SearchLast(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLast(100.0)!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLinear

Ищет элемент равный образцу в массиве.

```
int SearchLinear(
    float element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Примечание

Поиск производится последовательным перебором элементов массива (линейный поиск для несортированных массивов).

### Пример:

```
//--- example for CArrayFloat::SearchLinear(float)
#include <Arrays\ArrayFloat.mqh>
//---

void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- search element
    if(array.SearchLinear(100.0)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## Save

Сохраняет данные массива в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayFloat::Save(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    int          file_handle;
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add 100 arrays elements
    for(int i=0;i<100;i++)
    {
        array.Add(i);
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
        }
    }
}
```

```
    }
    FileClose(file_handle);
}
delete array;
}
```

## Load

Загружает данные массива из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayFloat::Load(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    int          file_handle;
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- use arrays elements
    for(int i=0;i<array.Total();i++)
```

```
{  
    printf("Element[%d] = %f", i, array.At(i));  
}  
delete array;  
}
```

## Type

Получает идентификатор типа массива.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа массива (для CArrayFloat - 87).

### Пример:

```
//--- example for CArrayFloat::Type()
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
{
    CArrayFloat *array=new CArrayFloat;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```

## CArrayDouble

Класс CArrayDouble является классом динамического массива переменных типа double.

### Описание

Класс CArrayDouble обеспечивает возможность работы с динамическим массивом переменных типа double. В классе реализованы возможности добавления/вставки/удаления элементов массива, сортировки массива, поисков в отсортированном массиве. Кроме того, реализованы методы работы с файлом.

### Декларация

```
class CArrayDouble : public CArray
```

### Заголовок

```
#include <Arrays\ArrayDouble.mqh>
```

### Иерархия наследования

```
CObject  
CArray  
CArrayDouble
```

### Прямые потомки

[CDoubleBuffer](#)

### Методы класса по группам

Атрибуты	
<a href="#">Delta</a>	Устанавливает величину допуска сравнения
<a href="#">Управление памятью</a>	
<a href="#">Reserve</a>	Резервирует память для увеличения размеров массива
<a href="#">Resize</a>	Устанавливает новый (меньший) размер массива
<a href="#">Shutdown</a>	Очищает массив с полным освобождением памяти
<a href="#">Наполнение</a>	
<a href="#">Add</a>	Добавляет элемент в конец массива
<a href="#">AddArray</a>	Добавляет в конец массива элементы из другого массива
<a href="#">AddArray</a>	Добавляет в конец массива элементы из другого массива

<a href="#"><u>Insert</u></a>	Вставляет элемент в массив в указанную позицию
<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<b>Изменение</b>	
<a href="#"><u>Update</u></a>	Изменяет элемент в указанной позиции массива
<a href="#"><u>Shift</u></a>	Перемещает элемент из указанной позиции массива на указанное смещение
<b>Удаление</b>	
<a href="#"><u>Delete</u></a>	Удаляет элемент из указанной позиции массива
<a href="#"><u>DeleteRange</u></a>	Удаляет группу элементов из указанной позиции массива
<b>Доступ</b>	
<a href="#"><u>At</u></a>	Получает элемент из указанной позиции массива
<b>Сравнение</b>	
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<b>Поиск экстремумов</b>	
<a href="#"><u>Minimum</u></a>	Получает индекс минимального значения в указанном диапазоне
<a href="#"><u>Maximum</u></a>	Получает индекс максимального значения в указанном диапазоне
<b>Операции с отсортированным массивом</b>	
<a href="#"><u>InsertSort</u></a>	Вставляет элемент в отсортированный массив
<a href="#"><u>Search</u></a>	Ищет элемент равный образцу в отсортированном массиве
<a href="#"><u>SearchGreat</u></a>	Ищет элемент больше образца в отсортированном массиве

<a href="#">SearchLess</a>	Ищет элемент меньше образца в сортированном массиве
<a href="#">SearchGreatOrEqual</a>	Ищет элемент больше или равный образцу в сортированном массиве
<a href="#">SearchLessOrEqual</a>	Ищет элемент меньше или равный образцу в сортированном массиве
<a href="#">SearchFirst</a>	Ищет первый элемент равный образцу в сортированном массиве
<a href="#">SearchLast</a>	Ищет последний элемент равный образцу в сортированном массиве
<a href="#">SearchLinear</a>	Ищет элемент равный образцу в массиве
<b>Ввод/вывод</b>	
<b>virtual <a href="#">Save</a></b>	Сохраняет данные массива в файле
<b>virtual <a href="#">Load</a></b>	Загружает данные массива из файла
<b>virtual <a href="#">Type</a></b>	Получает идентификатор типа массива

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

**Методы унаследованные от CArray**

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

## Delta

Устанавливает допуск сравнения.

```
void Delta(
    double delta      // допуск
)
```

### Параметры

*delta*  
 [in] новое значение допуска сравнения.

### Возвращаемое значение

Нет

### Примечание

Допуск сравнения используется при поисках. Значения считаются равными, если их разница меньше или равна допуску. По умолчанию допуск равен 0.0.

### Пример:

```
//--- example for CArrayDouble::Delta(double)
#include <Arrays\ArrayDouble.mqh>
//---

void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if (array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- set compare variation
    array.Delta(0.001);
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## Reserve

Резервирует память для увеличения размеров массива.

```
bool Reserve(
    int size        // количество
)
```

### Параметры

*size*

[in] Количество дополнительных элементов массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка запросить количество меньше или равное нулю, либо если массив увеличить не удалось.

### Примечание

Для уменьшения фрагментации памяти, приращение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayDouble::Reserve(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## Resize

Устанавливает новый (меньший) размер массива.

```
bool Resize(
    int size        // размер
)
```

### Параметры

*size*

[in] Новый размер массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка установить размер меньше нуля.

### Примечание

Изменение размера массива позволяет оптимально использовать память. Лишние элементы справа теряются. Для уменьшения фрагментации памяти, изменение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayDouble::Resize(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- resize array
    if(!array.Resize(10))
    {
        printf("Resize error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shutdown

Очищает массив с полным освобождением памяти.

```
bool Shutdown()
```

### Возвращаемое значение

true в случае успеха, false - если произошла ошибка.

### Пример:

```
//--- example for CArrayDouble::Shutdown()
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Add

Добавляет элемент в конец массива.

```
bool Add(
    double element      // элемент для добавления
)
```

### Параметры

*element*

[in] Значение элемента для добавления в массив.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элемент.

### Пример:

```
//--- example for CArrayDouble::Add(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    for(int i=0;i<100;i++)
    {
        if(!array.Add(i))
        {
            printf("Element addition error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const double& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayDouble::AddArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const CArrayDouble* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayDouble-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayDouble::AddArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayDouble *src=new CArrayDouble;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Insert

Вставляет элемент в массив в указанную позицию.

```
bool Insert(
    double element,      // элемент для вставки
    int     pos          // позиция
)
```

### Параметры

*element*

[in] Значение элемента для вставки в массив

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayDouble::Insert(double,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert elements
    for(int i=0;i<100;i++)
    {
        if(!array.Insert(i,0))
        {
            printf("Insert error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    const double& src[],      // массив-источник
    int          pos         // позиция
)
```

### Параметры

*src[]*  
 [in] Ссылка на массив-источник элементов для вставки

*pos*  
 [in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayDouble::InsertArray(const double &[],int)
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    CArrayDouble* src,           // указатель на источник
    int          pos            // позиция
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayDouble-источник элементов для вставки.

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayDouble::InsertArray(const CArrayDouble*,int)
#include <Arrays\ArrayDouble.mqh>
//---

void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayDouble *src=new CArrayDouble;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete src;
    }
}
```

```
    delete array;
    return;
}
//--- delete source array
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const double& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayDouble::AssignArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const CArrayDouble* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayDouble-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayDouble::AssignArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---

void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayDouble *src =new CArrayDouble;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- arrays is identical  
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Update

Изменяет элемент в указанной позиции массива.

```
bool Update(
    int    pos,           // позиция
    double element        // значение
)
```

### Параметры

*pos*  
 [in] Позиция элемента в массиве для изменения

*element*  
 [in] Новое значение элемента

### Возвращаемое значение

true в случае успеха, false - если нет возможности изменить элемент.

### Пример:

```
//--- example for CArrayDouble::Update(int,double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- update element
    if(!array.Update(0,100.0))
    {
        printf("Update error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shift

Перемещает элемент из указанной позиции массива на указанное смещение.

```
bool Shift(
    int pos,           // позиция
    int shift          // смещение
)
```

### Параметры

*pos*

[in] Позиция перемещаемого элемента массиве

*shift*

[in] Значение смещения (как положительное, так и отрицательное значение).

### Возвращаемое значение

true в случае успеха, false - если нет возможности переместить элемент.

### Пример:

```
//--- example for CArrayDouble::Shift(int,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shift element
    if(!array.Shift(10,-5))
    {
        printf("Shift error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Delete

Удаляет элемент из указанной позиции массива.

```
bool Delete(  
    int pos        // позиция  
)
```

### Параметры

*pos*

[in] Позиция удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элемент.

### Пример:

```
//--- example for CArrayDouble::Delete(int)  
#include <Arrays\ArrayDouble.mqh>  
//---  
void OnStart()  
{  
    CArrayDouble *array=new CArrayDouble;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```

## DeleteRange

Удаляет группу элементов из указанной позиции массива.

```
bool DeleteRange(
    int   from,      // позиция первого элемента
    int   to        // позиция последнего элемента
)
```

### Параметры

*from*

[in] Позиция первого удаляемого элемента в массиве.

*to*

[in] Позиция последнего удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элементы.

### Пример:

```
//--- example for CArrayDouble::DeleteRange(int,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- delete elements
    if(!array.DeleteRange(0,10))
    {
        printf("Delete error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## At

Получает элемент из указанной позиции массива.

```
double At(
    int pos        // позиция
) const
```

### Параметры

*pos*

[in] Позиция искомого элемента в массиве.

### Возвращаемое значение

Значение элемента в случае успеха, DBL\_MAX если была попытка получить элемент из не существующей позиции (при этом код последней ошибки ERR\_OUT\_OF\_RANGE).

### Примечание

Разумеется, DBL\_MAX может быть и валидным значением элемента массива, поэтому, получив такое значение, всегда проверяйте код последней ошибки.

### Пример:

```
//--- example for CArrayDouble::At(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    for(int i=0;i<array.Total();i++)
    {
        double result=array.At(i);
        if(result==DBL_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
            //--- ошибка чтения из массива
            printf("Get element error");
            delete array;
            return;
        }
    }
    //--- use element
```

```
//--- . . .
}
//--- delete array
delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const double& src[]          // массив-источник
) const
```

### Параметры

*src []*  
[in] Ссылка на массив-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayDouble::CompareArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete array
    delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const CArrayDouble* src          // указатель на источник
) const
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayDouble-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayDouble::CompareArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---

void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayDouble *src=new CArrayDouble;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete arrays
    delete src;
    delete array;
}
```

## Minimum

Получает индекс минимального элемента массива в указанном интервале.

```
int Minimum(  
    int start,      // стартовый индекс  
    int count       // количество  
) const
```

### Параметры

*start*

[in] Стартовый индекс интервала поиска.

*count*

[in] Размер интервала поиска (количество элементов).

### Возвращаемое значение

Индекс минимального элемента в указанном интервале.

## Maximum

Получает индекс максимального элемента массива в указанном интервале.

```
int Maximum(
    int start,           // стартовый индекс
    int count            // количество
) const
```

### Параметры

*start*

[in] Стартовый индекс интервала поиска.

*count*

[in] Размер интервала поиска (количество элементов).

### Возвращаемое значение

Индекс максимального элемента в указанном интервале.

## InsertSort

Вставляет элемент в сортированный массив.

```
bool InsertSort(
    double element      // элемент для вставки
)
```

### Параметры

*element*

[in] Значение элемента для вставки в сортированный массив

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayDouble::InsertSort(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- insert element
    if(!array.InsertSort(100.0))
    {
        printf("Insert error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Search

Ищет элемент равный образцу в сортированном массиве.

```
int Search(
    double element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayDouble::Search(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.Search(100.0)!=-1) printf("Element found");
    else                         printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreat

Ищет элемент больше образца в сортированном массиве.

```
int SearchGreat(
    double element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayDouble::SearchGreat(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreat(100.0)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLess

Ищет элемент меньше образца в сортированном массиве.

```
int SearchLess(
    double element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayDouble:: SearchLess(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLess(100.0)!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreatOrEqual

Ищет элемент больше или равный образцу в сортированном массиве.

```
int SearchGreatOrEqual(
    double element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayDouble::SearchGreatOrEqual(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreatOrEqual(100.0)!=-1) printf("Element found");
    else                                     printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLessOrEqual

Ищет элемент меньше или равный образцу в сортированном массиве.

```
int SearchLessOrEqual(
    double element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayDouble::SearchLessOrEqual(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLessOrEqual(100.0)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchFirst

Ищет первый элемент равный образцу в сортированном массиве.

```
int SearchFirst(
    double element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayDouble::SearchFirst(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchFirst(100.0)!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLast

Ищет последний элемент равный образцу в сортированном массиве.

```
int SearchLast(
    double element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayDouble::SearchLast(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLast(100.0)!=-1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLinear

Ищет элемент равный образцу в массиве.

```
int SearchLinear(
    double element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Примечание

Поиск производится последовательным перебором элементов массива (линейный поиск для несорттированных массивов).

### Пример:

```
//--- example for CArrayDouble::SearchLinear(double)
#include <Arrays\ArrayDouble.mqh>
//---

void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- search element
    if(array.SearchLinear(100.0)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## Save

Сохраняет данные массива в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayDouble::Save(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    int          file_handle;
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add 100 arrays elements
    for(int i=0;i<100;i++)
    {
        array.Add(i);
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
        }
    }
}
```

```
    }
    FileClose(file_handle);
}
//--- delete array
delete array;
}
```

## Load

Загружает данные массива из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayDouble::Load(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    int          file_handle;
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- use arrays elements
    for(int i=0;i<array.Total();i++)
```

```
{  
    printf("Element[%d] = %f", i, array.At(i));  
}  
//--- delete array  
delete array;  
}
```

## Type

Получает идентификатор типа массива.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа массива (для CArrayDouble - 87).

### Пример:

```
//--- example for CArrayDouble::Type()
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
{
    CArrayDouble *array=new CArrayDouble;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```

## CArrayString

Класс CArrayString является классом динамического массива переменных типа string.

### Описание

Класс CArrayString обеспечивает возможность работы с динамическим массивом переменных типа string. В классе реализованы возможности добавления/вставки/удаления элементов массива, сортировки массива, поисков в отсортированном массиве. Кроме того, реализованы методы работы с файлом.

### Декларация

```
class CArrayString : public CArray
```

### Заголовок

```
#include <Arrays\ArrayString.mqh>
```

### Иерархия наследования

```
CObject  
CArray  
CArrayString
```

### Методы класса по группам

Управление памятью	
<a href="#"><u>Reserve</u></a>	Резервирует память для увеличения размеров массива
<a href="#"><u>Resize</u></a>	Устанавливает новый (меньший) размер массива
<a href="#"><u>Shutdown</u></a>	Очищает массив с полным освобождением памяти
Наполнение	
<a href="#"><u>Add</u></a>	Добавляет элемент в конец массива
<a href="#"><u>AddArray</u></a>	Добавляет в конец массива элементы из другого массива
<a href="#"><u>AddArray</u></a>	Добавляет в конец массива элементы из другого массива
<a href="#"><u>Insert</u></a>	Вставляет элемент в массив в указанную позицию
<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции

<a href="#"><u>InsertArray</u></a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<a href="#"><u>AssignArray</u></a>	Копирует в массив элементы из другого массива
<b>Изменение</b>	
<a href="#"><u>Update</u></a>	Изменяет элемент в указанной позиции массива
<a href="#"><u>Shift</u></a>	Перемещает элемент из указанной позиции массива на указанное смещение
<b>Удаление</b>	
<a href="#"><u>Delete</u></a>	Удаляет элемент из указанной позиции массива
<a href="#"><u>DeleteRange</u></a>	Удаляет группу элементов из указанной позиции массива
<b>Доступ</b>	
<a href="#"><u>At</u></a>	Получает элемент из указанной позиции массива
<b>Сравнение</b>	
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<a href="#"><u>CompareArray</u></a>	Сравнивает массив с другим массивом
<b>Операции с отсортированным массивом</b>	
<a href="#"><u>InsertSort</u></a>	Вставляет элемент в отсортированный массив
<a href="#"><u>Search</u></a>	Ищет элемент равный образцу в отсортированном массиве
<a href="#"><u>SearchGreat</u></a>	Ищет элемент больше образца в отсортированном массиве
<a href="#"><u>SearchLess</u></a>	Ищет элемент меньше образца в отсортированном массиве
<a href="#"><u>SearchGreatOrEqual</u></a>	Ищет элемент больше или равный образцу в отсортированном массиве
<a href="#"><u>SearchLessOrEqual</u></a>	Ищет элемент меньше или равный образцу в отсортированном массиве
<a href="#"><u>SearchFirst</u></a>	Ищет первый элемент равный образцу в отсортированном массиве

<a href="#">SearchLast</a>	Ищет последний элемент равный образцу в сортированном массиве
<a href="#">SearchLinear</a>	Ищет элемент равный образцу в массиве
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Save</a>	Сохраняет данные массива в файле
<a href="#">virtual Load</a>	Загружает данные массива из файла
<a href="#">virtual Type</a>	Получает идентификатор типа массива

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

## Reserve

Резервирует память для увеличения размеров массива.

```
bool Reserve(
    int size        // количество
)
```

### Параметры

*size*

[in] Количество дополнительных элементов массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка запросить количество меньше или равное нулю, либо если массив увеличить не удалось.

### Примечание

Для уменьшения фрагментации памяти, приращение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayList::Reserve(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- reserve memory
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## Resize

Устанавливает новый (меньший) размер массива.

```
bool Resize(
    int size        // размер
)
```

### Параметры

*size*

[in] Новый размер массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка установить размер меньше нуля.

### Примечание

Изменение размера массива позволяет оптимально использовать память. Лишние элементы справа теряются. Для уменьшения фрагментации памяти, изменение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayList::Resize(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- resize array
    if(!array.Resize(10))
    {
        printf("Resize error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shutdown

Очищает массив с полным освобождением памяти.

```
bool Shutdown()
```

### Возвращаемое значение

true в случае успеха, false - если произошла ошибка.

### Пример:

```
//--- example for CArrayList::Shutdown()
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Add

Добавляет элемент в конец массива.

```
bool Add(
    string element      // элемент для добавления
)
```

### Параметры

*element*

[in] Значение элемента для добавления в массив.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элемент.

### Пример:

```
//--- example for CArrayList::Add(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    for(int i=0;i<100;i++)
    {
        if(!array.Add(IntegerToString(i)))
        {
            printf("Element addition error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const string& src[]          // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayList::AddArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const CArrayList* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayList - источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Пример:

```
//--- example for CArrayList::AddArray(const CArrayList*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayList *src=new CArrayList;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- add another array
    if(!array.AddArray(src))
    {
        printf("Array addition error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Insert

Вставляет элемент в массив в указанную позицию.

```
bool Insert(
    string element,      // элемент для вставки
    int     pos          // позиция
)
```

### Параметры

*element*

[in] Значение элемента для вставки в массив

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayList::Insert(string,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert elements
    for(int i=0;i<100;i++)
    {
        if(!array.Insert(IntegerToString(i),0))
        {
            printf("Insert error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    const string& src[],      // массив-источник
    int          pos         // позиция
)
```

### Параметры

*src[]*  
 [in] Ссылка на массив-источник элементов для вставки

*pos*  
 [in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayList::InsertArray(const string &[],int)
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    CArrayListString* src,           // указатель на источник
    int pos                         // позиция
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayListString - источник элементов для вставки.

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Пример:

```
//--- example for CArrayListString::InsertArray(const CArrayListString*,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayListString *array=new CArrayListString;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayListString *src=new CArrayListString;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- insert another array
    if(!array.InsertArray(src,0))
    {
        printf("Array inserting error");
        delete src;
    }
}
```

```
    delete array;
    return;
}
//--- delete source array
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const string& src[]           // массив-источник
)
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayList::AssignArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const CArrayListString* src      // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayListString - источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Пример:

```
//--- example for CArrayListString::AssignArray(const CArrayListString*)
#include <Arrays\ArrayString.mqh>
//---

void OnStart()
{
    CArrayListString *array=new CArrayListString;
    //---

    if(array==NULL)
    {
        printf("Object create error");
        return;
    }

    //--- create source array
    CArrayListString *src =new CArrayListString;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }

    //--- add source arrays elements
    //--- . . .
    //--- assign another array
    if(!array.AssignArray(src))
    {
        printf("Array assigned error");
        delete src;
        delete array;
        return;
    }
}
```

```
//--- arrays is identical  
//--- delete source array  
delete src;  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Update

Изменяет элемент в указанной позиции массива.

```
bool Update(
    int    pos,           // позиция
    string element        // значение
)
```

### Параметры

*pos*  
 [in] Позиция элемента в массиве для изменения

*element*  
 [in] Новое значение элемента

### Возвращаемое значение

true в случае успеха, false - если нет возможности изменить элемент.

### Пример:

```
//--- example for CArrayList::Update(int, string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- update element
    if(!array.Update(0, "ABC"))
    {
        printf("Update error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Shift

Перемещает элемент из указанной позиции массива на указанное смещение.

```
bool Shift(
    int pos,           // позиция
    int shift          // смещение
)
```

### Параметры

*pos*

[in] Позиция перемещаемого элемента массиве

*shift*

[in] Значение смещения (как положительное, так и отрицательное значение).

### Возвращаемое значение

true в случае успеха, false - если нет возможности переместить элемент.

### Пример:

```
//--- example for CArrayList::Shift(int,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shift element
    if(!array.Shift(10,-5))
    {
        printf("Shift error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Delete

Удаляет элемент из указанной позиции массива.

```
bool Delete(  
    int pos // позиция  
)
```

### Параметры

*pos*

[in] Позиция удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элемент.

### Пример:

```
//--- example for CArrayList::Delete(int)  
#include <Arrays\ArrayString.mqh>  
//---  
void OnStart()  
{  
    CArrayList *array=new CArrayList;  
    //---  
    if(array==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add arrays elements  
    //--- . . .  
    //--- delete element  
    if(!array.Delete(0))  
    {  
        printf("Delete error");  
        delete array;  
        return;  
    }  
    //--- delete array  
    delete array;  
}
```

## DeleteRange

Удаляет группу элементов из указанной позиции массива.

```
bool DeleteRange(
    int   from,      // позиция первого элемента
    int   to        // позиция последнего элемента
)
```

### Параметры

*from*

[in] Позиция первого удаляемого элемента в массиве.

*to*

[in] Позиция последнего удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элементы.

### Пример:

```
//--- example for CArrayList::DeleteRange(int,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- delete elements
    if(!array.DeleteRange(0,10))
    {
        printf("Delete error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## At

Получает элемент из указанной позиции массива.

```
string At(
    int pos        // позиция
) const
```

### Параметры

*pos*

[in] Позиция искомого элемента в массиве.

### Возвращаемое значение

Значение элемента в случае успеха, ""- если была попытка получить элемент из не существующей позиции (при этом код последней ошибки ERR\_OUT\_OF\_RANGE).

### Примечание

Разумеется, "" может быть и валидным значением элемента массива, поэтому, получив такое значение, всегда проверяйте код последней ошибки.

### Пример:

```
//--- example for CArrayList::At(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    for(int i=0;i<array.Total();i++)
    {
        string result=array.At(i);
        if(result=="" && GetLastError()==ERR_OUT_OF_RANGE)
        {
            //--- ошибка чтения из массива
            printf("Get element error");
            delete array;
            return;
        }
    }
    //--- use element
```

```
//--- . . .
}
//--- delete array
delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const string& src[]          // массив-источник
) const
```

### Параметры

*src []*

[in] Ссылка на массив-источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayList::CompareArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete array
    delete array;
}
```

## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArrays(
    const CArrayListString* src          // указатель на источник
) const
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayListString - источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayListString::CompareArray(const CArrayListString*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayListString *array=new CArrayListString;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayListString *src=new CArrayListString;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- add source arrays elements
    //--- . . .
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete arrays
    delete src;
    delete array;
}
```

## InsertSort

Вставляет элемент в сортированный массив.

```
bool InsertSort(
    string element      // элемент для вставки
)
```

### Параметры

*element*

[in] Значение элемента для вставки в сортированный массив

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Пример:

```
//--- example for CArrayList::InsertSort(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- insert element
    if(!array.InsertSort("ABC"))
    {
        printf("Insert error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Search

Ищет элемент равный образцу в сортированном массиве.

```
int Search(
    string element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayList::Search(string)
#include <Arrays\ArrayString.mqh>
//---

void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.Search("ABC")!=1) printf("Element found");
    else                      printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreat

Ищет элемент больше образца в сортированном массиве.

```
int SearchGreat(
    string element      // образец
) const
```

### Параметры

*element*  
 [in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayList::SearchGreat(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreat("ABC")!=1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLess

Ищет элемент меньше образца в сортированном массиве.

```
int SearchLess(
    string element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayList:: SearchLess(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLess("ABC") !=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchGreatOrEqual

Ищет элемент больше или равный образцу в сортированном массиве.

```
int SearchGreatOrEqual(
    string element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayList:: SearchGreatOrEqual(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchGreatOrEqual("ABC") !=-1) printf("Element found");
    else                                     printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLessOrEqual

Ищет элемент меньше или равный образцу в сортированном массиве.

```
int SearchLessOrEqual(
    string element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayList:: SearchLessOrEqual(string)
#include <Arrays\ArrayString.mqh>
//---

void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLessOrEqual("ABC")!=1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchFirst

Ищет первый элемент равный образцу в сортированном массиве.

```
int SearchFirst(
    string element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayList:: SearchFirst(string)
#include <Arrays\ArrayString.mqh>
//---

void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchFirst("ABC")!=1) printf("Element found");
    else                            printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLast

Ищет последний элемент равный образцу в сортированном массиве.

```
int SearchLast(
    string element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayList:: SearchLast(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- search element
    if(array.SearchLast("ABC") !=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## SearchLinear

Ищет элемент равный образцу в массиве.

```
int SearchLinear(
    string element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Примечание

Поиск производится последовательным перебором элементов массива (линейный поиск для несортированных массивов).

### Пример:

```
//--- example for CArrayList::SearchLinear(string)
#include <Arrays\ArrayString.mqh>
//---

void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- search element
    if(array.SearchLinear("ABC")!=1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
    delete array;
}
```

## Save

Сохраняет данные массива в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayList::Save(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    int          file_handle;
    CArrayList *array=new CArrayList;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add 100 arrays elements
    for(int i=0;i<100;i++)
    {
        array.Add(IntegerToString(i));
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
        }
    }
}
```

```
    }
    FileClose(file_handle);
}
delete array;
}
```

## Load

Загружает данные массива из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayList::Load(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    int file_handle;
    CArrayList *array=new CArrayList;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- use arrays elements
    for(int i=0;i<array.Total();i++)
```

```
{  
    printf("Element[%d] = '%s'", i, array.At(i));  
}  
delete array;  
}
```

## Type

Получает идентификатор типа массива.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа массива (для CArrayList - 89).

### Пример:

```
//--- example for CArrayList::Type()
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
{
    CArrayList *array=new CArrayList;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```

## CArrayObj

Класс CArrayObj является классом динамического массива указателей на экземпляры класса CObject и его наследников.

### Описание

Класс CArrayObj обеспечивает возможность работы с динамическим массивом указателей на экземпляры класса [CObject](#) и его наследников. Это дает возможность работы как с многомерными динамическими массивами примитивных типов данных, так и с более сложно организованными структурами данных.

В классе реализованы возможности добавления/вставки/удаления элементов массива, сортировки массива, поисков в сортированном массиве. Кроме того, реализованы методы работы с файлом.

Существуют определенные [тонкости работы](#) с классом CArrayObj.

### Декларация

```
class CArrayObj : public CArray
```

### Заголовок

```
#include <Arrays\ArrayObj.mqh>
```

### Иерархия наследования

```
CObject  
CArray  
CArrayObj
```

### Прямые потомки

```
CIndicators, CSeries
```

### Методы класса по группам

Атрибуты	
<a href="#">FreeMode</a>	Получает флаг управления памятью
<a href="#">FreeMode</a>	Устанавливает флаг управления памятью
<b>Управление памятью</b>	
<a href="#">Reserve</a>	Резервирует память для увеличения размеров массива
<a href="#">Resize</a>	Устанавливает новый (меньший) размер массива
<a href="#">Shutdown</a>	Очищает массив с полным освобождением памяти массива (но не элементов).

<b>Создание нового элемента</b>	
virtual <a href="#">CreateElement</a>	Создает новый элемент массива в указанной позиции.
<b>Наполнение</b>	
<a href="#">Add</a>	Добавляет элемент в конец массива
<a href="#">AddArray</a>	Добавляет в конец массива элементы из другого массива
<a href="#">Insert</a>	Вставляет элемент в массив в указанную позицию
<a href="#">InsertArray</a>	Вставляет в массив элементы из другого массива с указанной позиции
<a href="#">AssignArray</a>	Копирует в массив элементы из другого массива
<b>Изменение</b>	
<a href="#">Update</a>	Изменяет элемент в указанной позиции массива
<a href="#">Shift</a>	Перемещает элемент из указанной позиции массива на указанное смещение
<b>Удаление</b>	
<a href="#">Detach</a>	Получает элемент из указанной позиции с удалением его из массива
<a href="#">Delete</a>	Удаляет элемент из указанной позиции массива
<a href="#">DeleteRange</a>	Удаляет группу элементов из указанной позиции массива
<a href="#">Clear</a>	Удаляет все элементы массива без освобождения памяти массива
<b>Доступ</b>	
<a href="#">At</a>	Получает элемент из указанной позиции массива
<b>Сравнение</b>	
<a href="#">CompareArray</a>	Сравнивает массив с другим массивом
<b>Операции с отсортированным массивом</b>	
<a href="#">InsertSort</a>	Вставляет элемент в отсортированный массив
<a href="#">Search</a>	Ищет элемент равный образцу в отсортированном массиве

<a href="#">SearchGreat</a>	Ищет элемент больше образца в сортированном массиве
<a href="#">SearchLess</a>	Ищет элемент меньше образца в сортированном массиве
<a href="#">SearchGreatOrEqual</a>	Ищет элемент больше или равный образцу в сортированном массиве
<a href="#">SearchLessOrEqual</a>	Ищет элемент меньше или равный образцу в сортированном массиве
<a href="#">SearchFirst</a>	Ищет первый элемент равный образцу в сортированном массиве
<a href="#">SearchLast</a>	Ищет последний элемент равный образцу в сортированном массиве
<b>Ввод/вывод</b>	
<a href="#">virtual Save</a>	Сохраняет данные массива в файле
<a href="#">virtual Load</a>	Загружает данные массива из файла
<a href="#">virtual Type</a>	Получает идентификатор типа массива

**Методы унаследованные от CObject**Prev, Prev, Next, Next, [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

Практическое применение имеют массивы потомков класса CObject (в том числе все классы стандартной библиотеки).

Для примера рассмотрим вариант реализации двухмерного массива:

```
#include <Arrays\ArrayDouble.mqh>
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    int i,j;
    int first_size=10;
    int second_size=100;
//--- create array
    CArrayObj      *array=new CArrayObj;
    CArrayDouble   *sub_array;
//---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
}
```

```

    }

//--- create subarrays
for(i=0;i<first_size;i++)
{
    sub_array=new CArrayDouble;
    if(sub_array==NULL)
    {
        delete array;
        printf("Object create error");
        return;
    }
//--- fill array
for(j=0;j<second_size;j++)
{
    sub_array.Add(i*j);
}
array.Add(sub_array);
}

//--- create array OK
for(i=0;i<first_size;i++)
{
    sub_array=array.At(i);
    for(j=0;j<second_size;j++)
    {
        double element=sub_array.At(j);
        //--- use array element
    }
}
delete array;
}

```

## Тонкости работы

В классе реализован механизм управления динамической памятью, поэтому нужно быть очень внимательным при работе с элементами массива.

Механизм управления памятью можно включать/отключать при помощи метода `FreeMode(bool)`. По умолчанию механизм включен.

Соответственно, есть два варианта работы с классом `CArrayObj`:

1. Механизм управления памятью включен. (по умолчанию)

В этом случае, `CArrayObj` берет на себя ответственность за освобождение памяти элементов после их удаления из массива. При этом программа пользователя не должна освобождать элементы массива.

### Пример использования:

```

int i;
//--- создаем массив
CArrayObj *array=new CArrayObj;
//--- наполняем массив элементами
for(i=0;i<10;i++) array.Add(new CObject);
//--- что-то делаем
for(i=0;i<array.Total();i++)
{
    CObject *object=array.At(i);
    //--- действия с элементом
    . .
}
//--- удаляем массив вместе с элементами
delete array;

```

## 2. Механизм управления памятью выключен.

В этом случае, CArrayObj не отвечает за освобождение памяти элементов после их удаления из массива. При этом программа пользователя должна освобождать элементы массива.

### Пример использования:

```

int i;
//--- создаем массив
CArrayObj *array=new CArrayObj;
//--- выключаем механизм управления памятью
array.FreeMode(false);
//--- наполняем массив элементами
for(i=0;i<10;i++) array.Add(new CObject);
//--- что-то делаем
for(i=0;i<array.Total();i++)
{
    CObject *object=array.At(i);
    //--- действия с элементом
    . .
}
//--- удаляем элементы массива
while(array.Total()) delete array.Detach();
//--- удаляем пустой массив
delete array;

```

## FreeMode

Получает флаг управления памятью.

```
bool FreeMode() const
```

### Возвращаемое значение

Флаг управления памятью.

### Пример:

```
//--- example for CArrayObj::FreeMode()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get free mode flag
    bool array_free_mode=array.FreeMode();
    //--- delete array
    delete array;
}
```

## FreeMode

Устанавливает флаг управления памятью.

```
void FreeMode(
    bool mode        // новый флаг
)
```

### Параметры

*mode*

[in] Новое значение флага управления памятью.

### Возвращаемое значение

Нет.

### Примечание

Установка флага управления памятью - важный момент в использовании класса CArrayObj. Так как элементами массива являются указатели на динамические объекты, важно определить, что делать с ними при удалении из массива.

Если флаг установлен, то при удалении элемента из массива, элемент автоматически удаляется оператором `delete`. Если же флаг не установлен, то подразумевается, что указатель на удаляемый объект остается еще где-то в программе пользователя и будет освобожден ею (программой) впоследствии.

Если программа пользователя сбрасывает флаг управления памятью, пользователь должен понимать свою ответственность за удаление элементов массива перед завершением программы, так как в противном случае остается неосвобожденной память, занятая элементами при их создании оператором `new`.

При больших объемах данных, это может привести, в конце концов, даже к нарушению работоспособности терминала. Если программа пользователя не сбрасывает флаг управления памятью, существует другой "подводный камень".

Использование указателей-элементов массива, сохраненных где-нибудь в локальных переменных, после удаления массива, приведет к критической ошибке и аварийному завершению программы пользователя. По умолчанию флаг управления памятью установлен, то есть класс массива сам отвечает за освобождение памяти элементов.

### Пример:

```
//--- example for CArrayObj::FreeMode(bool)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if (array==NULL)
```

```
{  
    printf("Object create error");  
    return;  
}  
//--- reset free mode flag  
array.FreeMode(false);  
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## Reserve

Резервирует память для увеличения размеров массива.

```
bool Reserve(
    int size        // количество
)
```

### Параметры

*size*

[in] Количество дополнительных элементов массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка запросить количество меньше или равное нулю, либо если массив увеличить не удалось.

### Примечание

Для уменьшения фрагментации памяти, приращение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayObj::Reserve(int)
#include <Arrays\ArrayObj.mqh>
//---

void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    if(!array.Reserve(1024))
    {
        printf("Reserve error");
        delete array;
        return;
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
}
```

## Resize

Устанавливает новый (меньший) размер массива.

```
bool Resize(
    int size        // размер
)
```

### Параметры

*size*

[in] Новый размер массива.

### Возвращаемое значение

true в случае успеха, false - если была попытка установить размер меньше нуля.

### Примечание

Изменение размера массива позволяет оптимально использовать память. Лишние элементы справа теряются. Память утерянных элементов освобождается или нет в зависимости от режима управления памятью.

Для уменьшения фрагментации памяти, изменение размеров массива производится с шагом ранее заданным через метод Step(int), либо 16 (по умолчанию).

### Пример:

```
//--- example for CArrayObj::Resize(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- resize array
    if(!array.Resize(10))
    {
        printf("Resize error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
```

{}

## Clear

Удаляет все элементы массива без освобождения памяти массива.

```
void Clear()
```

### Возвращаемое значение

Нет.

### Примечание

Если включен механизм управления памятью, память удаляемых элементов освобождается.

### Пример:

```
//--- example for CArrayObj::Clear()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- clear array
    array.Clear();
    //--- delete array
    delete array;
}
```

## Shutdown

Очищает массив с полным освобождением памяти массива (но не элементов).

```
bool Shutdown()
```

### Возвращаемое значение

true в случае успеха, false - если произошла ошибка.

### Примечание

Если включен механизм управления памятью, память удаляемых элементов освобождается.

### Пример:

```
//--- example for CArrayObj::Shutdown()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shutdown array
    if(!array.Shutdown())
    {
        printf("Shutdown error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## CreateElement

Создает новый элемент массива в указанной позиции.

```
bool CreateElement(
    int index      // позиция
)
```

### Параметры

*index*

[in] Позиция, в которой нужно создать новый элемент.

### Возвращаемое значение

true в случае успеха, false - если нет возможности создать элемент.

### Примечание

Метод CreateElement(int) в классе CArrayObj всегда возвращает false и не выполняет каких-либо действий. При необходимости в классе-наследнике, метод CreateElement(int) должен быть реализован.

### Пример:

```
//--- example for CArrayObj::CreateElement(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    int size=100;
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- fill array
    array.Reserve(size);
    for(int i=0;i<size;i++)
    {
        if(!array.CreateElement(i))
        {
            printf("Element create error");
            delete array;
            return;
        }
    }
    //--- use array
```

```
//--- . . .
//--- delete array
delete array;
}
```

## Add

Добавляет элемент в конец массива.

```
bool Add(
    CObject* element      // элемент для добавления
)
```

### Параметры

*element*  
 [in] Значение элемента для добавления в массив.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элемент.

### Примечание

Элемент не добавится в массив, если в качестве параметра передать некорректный указатель (например NULL).

### Пример:

```
//--- example for CArrayObj::Add(CObject*)
#include <Arrays\ArrayObj.mqh>
//---

void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---

    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add 100 arrays elements
    for(int i=0;i<100;i++)
    {
        if(!array.Add(new CObject))
        {
            printf("Element addition error");
            delete array;
            return;
        }
    }
    //--- use array
    //--- . . .
    //--- delete array
    delete array;
```

{}

## AddArray

Добавляет в конец массива элементы из другого массива.

```
bool AddArray(
    const CArrayObj * src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса [CArrayDouble](#) - источник элементов для добавления.

### Возвращаемое значение

true в случае успеха, false - если нет возможности добавить элементы.

### Примечание

Добавление элементов из массива в массив фактически является копированием указателей. В связи с этим, при вызове метода, существует "подводный камень" - возможность существования указателя на динамический объект более чем в одной переменной.

```
---- example
extern bool      make_error;
extern int       error;
extern CArrayObj *src;
---- создаем новый экземпляр CArrayObj
---- по умолчанию механизм управления памятью в нем включен
CArrayObj *array=new CArrayObj;
---- добавляем (копируем) элементы из массива-источника
if(array!=NULL)
    bool result=array.AddArray(src);
if(make_error)
{
    ---- совершаём ошибочные действия
    switch(error)
    {
        case 0:
            ---- удаляем массив-источник, не проверив его флаг управления памятью
            delete src;
            ---- результат:
            ---- возможно обращение
            ---- по "битому" указателю в массиве-приемнике
            break;
        case 1:
            ---- выключаем механизм управления памятью в массиве-источнике
            if(src.FreeMode()) src.FreeMode(false);
            ---- но не удаляем массив-источник
            ---- результат:
            ---- после удаления массива-приемника, возможно обращение
```

```

//--- по "битому" указателю в массиве-источнике
break;

case 2:
//--- выключаем механизм управления памятью в массиве-источнике
src.FreeMode(false);
//--- выключаем механизм управления памятью в массиве-приемнике
array.FreeMode(false);
//--- результат:
//--- после завершения программы, получаем "утечку памяти"
break;
}

}

else
{
//--- выключаем механизм управления памятью в массиве-источнике
if(src.FreeMode()) src.FreeMode(false);
//--- удаляем массив-источник
delete src;
//--- результат:
//--- обращение к элементам массива-приемника будет корректным
//--- удаление массива-приемника приведет к удалению его элементов
}

```

**Пример:**

```

//--- example for CArrayObj::AddArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
CArrayObj *array=new CArrayObj;
//---
if(array==NULL)
{
printf("Object create error");
return;
}
//--- create source array
CArrayObj *src=new CArrayObj;
if(src==NULL)
{
printf("Object create error");
delete array;
return;
}

```

```
//--- reset free mode flag
src.FreeMode(false);
//--- fill source array
//...
//--- add another array
if(!array.AddArray(src))
{
    printf("Array addition error");
    delete src;
    delete array;
    return;
}
//--- delete source array without elements
delete src;
//--- use array
//...
//--- delete array
delete array;
}
```

## Insert

Вставляет элемент в массив в указанную позицию.

```
bool Insert(
    CObject* element,      // элемент для вставки
    int     pos            // позиция
)
```

### Параметры

*element*

[in] Значение элемента для вставки в массив

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Примечание

Элемент не добавится в массив, если в качестве параметра передать некорректный указатель (например NULL).

### Пример:

```
//--- example for CArrayObj::Insert(CObject*,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert elements
    for(int i=0;i<100;i++)
    {
        if(!array.Insert(new CObject,0))
        {
            printf("Insert error");
            delete array;
            return;
        }
    }
}
```

```
//--- use array  
//--- . . .  
//--- delete array  
delete array;  
}
```

## InsertArray

Вставляет в массив элементы из другого массива с указанной позиции.

```
bool InsertArray(
    const CArrayObj* src,      // указатель на источник
    int pos                   // позиция
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayObj-источник элементов для вставки.

*pos*

[in] Позиция в массиве для вставки

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элементы.

### Примечание

См. [CArrayObj::AddArray\(const CArrayObj\\*\).](#)

### Пример:

```
//--- example for CArrayObj::InsertArray(const CArrayObj*,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayObj *src=new CArrayObj;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- reset free mode flag
    src.FreeMode(false);
    //--- fill source array
```

```
//--- . . .
//--- insert another array
if(!array.InsertArray(src,0))
{
    printf("Array inserting error");
    delete src;
    delete array;
    return;
}
//--- delete source array without elements
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```

## AssignArray

Копирует в массив элементы из другого массива.

```
bool AssignArray(
    const CArrayObj* src          // указатель на источник
)
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayObj - источник элементов для копирования.

### Возвращаемое значение

true в случае успеха, false - если нет возможности скопировать элементы.

### Примечание

Если перед вызовом AssignArray массив приемник не пуст, все его элементы будут удалены из массива и, если установлен флаг управления памятью, память удаленных элементов будет освобождена. Массив-приемник становится точной копией массива источника. Дополнительно см. [CArrayObj::AddArray\(const CArrayObj\\*\)](#).

### Пример:

```
//--- example for CArrayObj::AssignArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayObj *src=new CArrayObj;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- reset free mode flag
    src.FreeMode(false);
    //--- fill source array
    //--- . . .
```

```
//--- assign another array
if(!array.AssignArray(src))
{
    printf("Array assigned error");
    delete src;
    delete array;
    return;
}
//--- arrays is identical
//--- delete source array without elements
delete src;
//--- use array
//--- . . .
//--- delete array
delete array;
}
```

## Update

Изменяет элемент в указанной позиции массива.

```
bool Update(
    int      pos,           // позиция
    CObject* element        // значение
)
```

### Параметры

*pos*  
 [in] Позиция элемента в массиве для изменения

*element*  
 [in] Новое значение элемента

### Возвращаемое значение

true в случае успеха, false - если нет возможности изменить элемент.

### Примечание

Элемент не изменится, если в качестве параметра передать некорректный указатель (например NULL). Если включен механизм управления памятью, память замещаемого элемента освобождается.

### Пример:

```
//--- example for CArrayObj::Update(int,CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- update element
    if(!array.Update(0,new CObject))
    {
        printf("Update error");
        delete array;
        return;
    }
}
```

```
//--- delete array  
delete array;  
}
```

## Shift

Перемещает элемент из указанной позиции массива на указанное смещение.

```
bool Shift(
    int pos,           // позиция
    int shift          // смещение
)
```

### Параметры

*pos*

[in] Позиция перемещаемого элемента массиве

*shift*

[in] Значение смещения (как положительное, так и отрицательное значение).

### Возвращаемое значение

true в случае успеха, false - если нет возможности переместить элемент.

### Пример:

```
//--- example for CArrayObj::Shift(int,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- shift element
    if(!array.Shift(10,-5))
    {
        printf("Shift error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## Detach

Изымаёт элемент из указанной позиции массива.

```
CObject* Detach(
    int pos        // позиция
)
```

### Параметры

*pos*

[in] Позиция изымаемого элемента в массиве.

### Возвращаемое значение

Указатель на изъятый элемент в случае успеха, NULL - если нет возможности изъять элемент.

### Примечание

При изъятии из массива элемент не удалится при любом состоянии флага управления памятью.

Указатель на изъятый из массива элемент необходимо освободить после использования.

### Пример:

```
//--- example for CArrayObj::Detach(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . .
    CObject *object=array.Detach(0);
    if(object==NULL)
    {
        printf("Detach error");
        delete array;
        return;
    }
    //--- use element
    //--- . .
    //--- delete element
    delete object;
    //--- delete array
```

```
    delete array;
}
```

## Delete

Удаляет элемент из указанной позиции массива.

```
bool Delete(
    int pos        // позиция
)
```

### Параметры

*pos*

[in] Позиция удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элемент.

### Примечание

Если включен механизм управления памятью, память удаляемых элементов освобождается.

### Пример:

```
//--- example for CArrayObj::Delete(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    if(!array.Delete(0))
    {
        printf("Delete error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
}
```

## DeleteRange

Удаляет группу элементов из указанной позиции массива.

```
bool DeleteRange(
    int from,           // позиция первого элемента
    int to              // позиция последнего элемента
)
```

### Параметры

*from*

[in] Позиция первого удаляемого элемента в массиве.

*to*

[in] Позиция последнего удаляемого элемента в массиве.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элементы.

### Примечание

Если включен механизм управления памятью, память удаляемых элементов освобождается.

### Пример:

```
//--- example for CArrayObj::DeleteRange(int,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- delete elements
    if(!array.DeleteRange(0,10))
    {
        printf("Delete error");
        delete array;
        return;
    }
    //--- delete array
    delete array;
```

{}

## At

Получает элемент из указанной позиции массива.

```
CObject* At(
    int pos        // позиция
)
```

### Параметры

*pos*

[in] Позиция искомого элемента в массиве.

### Возвращаемое значение

Значение элемента в случае успеха, NULL- если была попытка получить элемент из не существующей позиции.

### Пример:

```
//--- example for CArrayObj::At(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add elements
    //--- . . .
    for(int i=0;i<array.Total();i++)
    {
        CObject *result=array.At(i);
        if(result==NULL)
        {
            //--- ошибка чтения из массива
            printf("Get element error");
            delete array;
            return;
        }
        //--- use element
        //--- . . .
    }
    delete array;
}
```



## CompareArray

Сравнивает массив с другим массивом.

```
bool CompareArray(
    const CArrayObj* src          // указатель на источник
) const
```

### Параметры

*src*

[in] Указатель на экземпляр класса CArrayObj - источник элементов для сравнения.

### Возвращаемое значение

true в случае массивы равны, false - если нет.

### Пример:

```
//--- example for CArrayObj::CompareArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---

void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source array
    CArrayObj *src=new CArrayObj;
    if(src==NULL)
    {
        printf("Object create error");
        delete array;
        return;
    }
    //--- fill source array
    //--- . . .
    //--- compare with another array
    int result=array.CompareArray(src);
    //--- delete arrays
    delete src;
    delete array;
}
```

## InsertSort

Вставляет элемент в сортированный массив.

```
bool InsertSort(
    CObject* element      // элемент для вставки
)
```

### Параметры

*element*

[in] Значение элемента для вставки в сортированный массив

### Возвращаемое значение

true в случае успеха, false - если нет возможности вставить элемент.

### Примечание

Элемент не добавится в массив, если в качестве параметра передать некорректный указатель (например NULL).

### Пример:

```
//--- example for CArrayObj::InsertSort(CObject*)
#include <Arrays\ArrayObj.mqh>
//---

void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---

    if(array==NULL)
    {
        printf("Object create error");
        return;
    }

    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- insert element
    if(!array.InsertSort(new CObject))
    {
        printf("Insert error");
        delete array;
        return;
    }

    //--- delete array
    delete array;
}
```

## Search

Ищет элемент равный образцу в сортированном массиве.

```
int Search(
    CObject* element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayObj::Search(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- create sample
    CObject *sample=new CObject;
    if(sample==NULL)
    {
        printf("Sample create error");
        delete array;
        return;
    }
    //--- set sample attributes
    //--- . . .
    //--- search element
    if(array.Search(sample)!=-1) printf("Element found");
    else                         printf("Element not found");
    //--- delete array
```

```
    delete array;
}
```

## SearchGreat

Ищет элемент больше образца в сортированном массиве.

```
int SearchGreat(
    CObject* element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayObj::SearchGreat(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- create sample
    CObject *sample=new CObject;
    if(sample==NULL)
    {
        printf("Sample create error");
        delete array;
        return;
    }
    //--- set sample attributes
    //--- . . .
    //--- search element
    if(array.SearchGreat(sample)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
```

```
    delete array;
}
```

## SearchLess

Ищет элемент меньше образца в сортированном массиве.

```
int SearchLess(
    CObject* element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayObj:: SearchLess(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- create sample
    CObject *sample=new CObject;
    if(sample==NULL)
    {
        printf("Sample create error");
        delete array;
        return;
    }
    //--- set sample attributes
    //--- . . .
    //--- search element
    if(array.SearchLess(sample)!=-1) printf("Element found");
    else                           printf("Element not found");
    //--- delete array
```

```
    delete array;
}
```

## SearchGreaterEqual

Ищет элемент больше или равный образцу в сортированном массиве.

```
int SearchGreaterEqual(
    CObject* element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayObj::SearchGreaterEqual(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- create sample
    CObject *sample=new CObject;
    if(sample==NULL)
    {
        printf("Sample create error");
        delete array;
        return;
    }
    //--- set sample attributes
    //--- . . .
    //--- search element
    if(array.SearchGreaterEqual(sample)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
```

```
    delete array;
}
```

## SearchLessOrEqual

Ищет элемент меньше или равный образцу в сортированном массиве.

```
int SearchLessOrEqual(
    CObject* element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayObj:: SearchLessOrEqual(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- create sample
    CObject *sample=new CObject;
    if(sample==NULL)
    {
        printf("Sample create error");
        delete array;
        return;
    }
    //--- set sample attributes
    //--- . . .
    //--- search element
    if(array.SearchLessOrEqual(sample)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
```

```
    delete array;
}
```

## SearchFirst

Ищет первый элемент равный образцу в сортированном массиве.

```
int SearchFirst(
    CObject* element      // образец
) const
```

### Параметры

*element*  
 [in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayObj::SearchFirst(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- create sample
    CObject *sample=new CObject;
    if(sample==NULL)
    {
        printf("Sample create error");
        delete array;
        return;
    }
    //--- set sample attributes
    //--- . . .
    //--- search element
    if(array.SearchFirst(sample)!=-1) printf("Element found");
    else                                printf("Element not found");
    //--- delete array
```

```
    delete array;
}
```

## SearchLast

Ищет последний элемент равный образцу в сортированном массиве.

```
int SearchLast(
    CObject* element      // образец
) const
```

### Параметры

*element*

[in] Образец элемента для поиска в массиве.

### Возвращаемое значение

Позиция найденного элемента в случае успеха, -1 - если элемент не найден.

### Пример:

```
//--- example for CArrayObj:: SearchLast(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- sort array
    array.Sort();
    //--- create sample
    CObject *sample=new CObject;
    if(sample==NULL)
    {
        printf("Sample create error");
        delete array;
        return;
    }
    //--- set sample attributes
    //--- . . .
    //--- search element
    if(array.SearchLast(sample)!=-1) printf("Element found");
    else                           printf("Element not found");
    //--- delete array
```

```
    delete array;
}
```

## Save

Сохраняет данные массива в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого при помощи функции FileOpen(...) бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CArrayObj::Save(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    int file_handle;
    CArrayObj *array=new CArrayObj;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add arrays elements
    //--- . . .
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
}
```

```
    delete array;
}
```

## Load

Загружает данные массива из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Примечание

При чтении из файла элементов массива, для создания каждого элемента вызывается метод [CArrayObj::CreateElement\(int\)](#).

### Пример:

```
//--- example for CArrayObj::Load(int)
#include <Arrays\ArrayObj.mqh>
//---

void OnStart()
{
    int file_handle;
    CArrayObj *array=new CArrayObj;
    //---
    if(array!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!array.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete array;
            FileClose(file_handle);
            //---
            return;
        }
    }
}
```

```
    FileClose(file_handle);
}
//--- use arrays elements
//--- . . .
//--- delete array
delete array;
}
```

## Type

Получает идентификатор типа массива.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа массива (для CArrayObj - 7778).

### Пример:

```
//--- example for CArrayObj::Type()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
{
    CArrayObj *array=new CArrayObj;
    //---
    if(array==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get array type
    int type=array.Type();
    //--- delete array
    delete array;
}
```

## CList

Класс CList является классом динамического списка экземпляров класса CObject и его наследников.

### Описание

Класс CList обеспечивает возможность работы со списком экземпляров класса [CObject](#) и его наследников. В классе реализованы возможности добавления/вставки/удаления элементов списка, сортировки списка, поисков в сортированном списке. Кроме того, реализованы методы работы с файлом.

Существуют определенные тонкости работы с классом CList. В классе реализован механизм управления динамической памятью, поэтому нужно быть очень внимательным при работе с элементами списка.

[Тонкости работы](#) с механизмом управления памятью аналогичны описанным в [CArrayObj](#).

### Декларация

```
class CList : public CObject
```

### Заголовок

```
#include <Arrays\List.mqh>
```

### Иерархия наследования

[CObject](#)

CList

### Методы класса по группам

Атрибуты	
<a href="#">FreeMode</a>	Получает флаг управления памятью при удалении элементов списка.
<a href="#">FreeMode</a>	Устанавливает флаг управления памятью при удалении элементов списка
<a href="#">Total</a>	Получает количество элементов в списке
<a href="#">IsSorted</a>	Получает флаг сортированности списка
<a href="#">SortMode</a>	Получает вариант сортировки
<b>Создание нового элемента</b>	
<b>virtual <a href="#">CreateElement</a></b>	Создает новый элемент списка
<b>Наполнение</b>	
<a href="#">Add</a>	Добавляет элемент в конец списка

<a href="#"><u>Insert</u></a>	Вставляет элемент в список в указанную позицию
<b>Удаление</b>	
<a href="#"><u>DetachCurrent</u></a>	Изымаает элемент из текущей позиции списка, не удаляя его "физически"
<a href="#"><u>DeleteCurrent</u></a>	Удаляет элемент из текущей позиции списка
<a href="#"><u>Delete</u></a>	Удаляет элемент из указанной позиции списка
<a href="#"><u>Clear</u></a>	Удаляет все элементы списка
<b>Навигация</b>	
<a href="#"><u>IndexOf</u></a>	Получает индекс указанного элемента списка
<a href="#"><u>GetNodeAtIndex</u></a>	Получает элемент из указанной позиции списка
<a href="#"><u>GetFirstNode</u></a>	Получает первый элемент списка
<a href="#"><u>GetPrevNode</u></a>	Получает предыдущий элемент списка
<a href="#"><u>GetCurrentNode</u></a>	Получает текущий элемент списка
<a href="#"><u>GetNextNode</u></a>	Получает следующий элемент списка
<a href="#"><u>GetLastNode</u></a>	Получает последний элемент списка
<b>Перемещение элементов</b>	
<a href="#"><u>Sort</u></a>	Сортирует список
<a href="#"><u>MoveToIndex</u></a>	Перемещает текущий элемент списка в указанную позицию
<a href="#"><u>Exchange</u></a>	Меняет элементы списка местами
<b>Сравнение</b>	
<a href="#"><u>CompareList</u></a>	Сравнивает список с другим списком
<b>Поиск</b>	
<a href="#"><u>Search</u></a>	Ищет элемент, равный образцу, в сортированном списке
<b>Ввод/вывод</b>	
virtual <a href="#"><u>Save</u></a>	Сохраняет данные списка в файл
virtual <a href="#"><u>Load</u></a>	Загружает данные списка из файла
virtual <a href="#"><u>Type</u></a>	Получает идентификатор типа списка

**Методы унаследованные от CObject**Prev, Prev, Next, Next, [Compare](#)



## FreeMode

Получает флаг управления памятью при удалении элементов списка.

```
bool FreeMode() const
```

### Возвращаемое значение

Флаг управления памятью.

### Пример:

```
//--- example for CList::FreeMode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get free mode flag
    bool list_free_mode=list.FreeMode();
    //--- delete list
    delete list;
}
```

## FreeMode

Устанавливает флаг управления памятью при удалении элементов списка.

```
void FreeMode(
    bool mode      // новое значение
)
```

### Параметры

*mode*

[in] Новое значение флага управления памятью.

### Примечание

Установка флага управления памятью - важный момент в использовании класса `CList`. Так как элементами списка являются указатели на динамические объекты, важно определить, что делать с ними при удалении из списка.

Если флаг установлен, то при удалении элемента из списка, элемент автоматически удаляется оператором `delete`. Если же флаг не установлен, то подразумевается, что указатель на удаляемый объект остается еще где-то в программе пользователя и будет освобожден ею (программой) впоследствии.

Если программа пользователя сбрасывает флаг управления памятью, пользователь должен понимать свою ответственность за удаление элементов списка перед завершением программы, так как в противном случае остается еще где-то в программе пользователя и будет освобожден ею (программой) впоследствии.

При больших объемах данных, это может привести, в конце концов, даже к нарушению работоспособности терминала.

Если программа пользователя не сбрасывает флаг управления памятью, существует другой "подводный камень". Использование указателей-элементов списка, сохраненных где-нибудь в локальных переменных, после удаления списка, приведет к критической ошибке и аварийному завершению программы пользователя. По умолчанию флаг управления памятью установлен, то есть класс списка сам отвечает за освобождение памяти элементов.

### Пример:

```
//--- example for CList::FreeMode(bool)
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
}
```

```
//--- reset free mode flag  
list.FreeMode(false);  
//--- use list  
//--- . . .  
//--- delete list  
delete list;  
}
```

## Total

Получает количество элементов в списке.

```
int Total() const
```

### Возвращаемое значение

Количество элементов в списке.

### Пример:

```
//--- example for CList::Total()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check total
    int total=list.Total();
    //--- use list
    //--- ...
    //--- delete list
    delete list;
}
```

## IsSorted

Получает флаг сортированности списка.

```
bool IsSorted(
    int mode=0           // вариант сортировки
) const
```

### Параметры

*mode=0*

[in] Проверяемый вариант сортировки

### Возвращаемое значение

Флаг сортированности списка. Если список сортированный по указанному варианту - true, иначе false.

### Примечание

Флаг сортированности списка нельзя изменить непосредственно. Флаг устанавливается методом Sort(int) и сбрасывается любыми методами добавления/вставки.

### Пример:

```
//--- example for CList::IsSorted()
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check sorted
    if(list.IsSorted(0))
    {
        //--- use methods for sorted list
        //--- ...
    }
    //--- delete list
    delete list;
}
```

## SortMode

Получает вариант сортировки.

```
int SortMode() const
```

### Возвращаемое значение

Вариант сортировки, либо -1 если список не сортирован.

### Пример:

```
//--- example for CList::SortMode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- check sort mode
    int sort_mode=list.SortMode();
    //--- use list
    //--- ...
    //--- delete list
    delete list;
}
```

## CreateElement

Создает новый элемент списка.

```
CObject* CreateElement()
```

### Возвращаемое значение

Указатель на вновь созданный элемент в случае успеха, NULL - если нет возможности создать элемент.

### Примечание

Метод CreateElement() в классе CList всегда возвращает NULL и не выполняет каких-либо действий. При необходимости в классе-наследнике, метод CreateElement() должен быть реализован.

### Пример:

```
//--- example for CList::CreateElement(int)
#include <Arrays\List.mqh>
//---
void OnStart()
{
    int      size=100;
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- fill list
    for(int i=0;i<size;i++)
    {
        CObject *object=list.CreateElement();
        if(object==NULL)
        {
            printf("Element create error");
            delete list;
            return;
        }
        list.Add(object);
    }
    //--- use list
    //--- . . .
    //--- delete list
    delete list;
}
```

## Add

Добавляет элемент в конец списка.

```
int Add(
    CObject* element      // элемент для добавления
)
```

### Параметры

*element*

[in] значение элемента для добавления в список.

### Возвращаемое значение

Индекс добавленного элемента, либо -1 в случае ошибки.

### Примечание

Элемент не добавится в список, если в качестве параметра передать некорректный указатель (например NULL).

### Пример:

```
//--- example for CList::Add(Cobject*)
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add 100 elements
    for(int i=0;i<100;i++)
    {
        if(list.Add(new CObject)==-1)
        {
            printf("Element addition error");
            delete list;
            return;
        }
    }
    //--- use list
    //--- . . .
    //--- delete list
    delete list;
```

{}

## Insert

Вставляет элемент в список в указанную позицию.

```
int Insert(
    CObject* element,      // элемент для вставки
    int      pos           // позиция
)
```

### Параметры

*element*

[in] значение элемента для вставки в список

*pos*

[in] позиция в списке для вставки

### Возвращаемое значение

Индекс вставленного элемента, либо -1 в случае ошибки.

### Примечание

Элемент не добавится в список, если в качестве параметра передать некорректный указатель (например NULL).

### Пример:

```
//--- example for CList::Insert(CObject*,int)
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- insert 100 elements
    for(int i=0;i<100;i++)
    {
        if(list.Insert(new CObject,0)==-1)
        {
            printf("Element insert error");
            delete list;
            return;
        }
    }
}
```

```
//--- use list  
//--- . . .  
//--- delete list  
delete list;  
}
```

## DetachCurrent

Изымает элемент из текущей позиции списка не удаляя его "физически".

```
CObject* DetachCurrent()
```

### Возвращаемое значение

Указатель на изъятый элемент в случае успеха, NULL - если нет возможности изъять элемент.

### Примечание

При изъятии из списка, элемент не удалится при любом состоянии флага управления памятью. Указатель на изъятый из списка элемент необходимо освободить после использования.

### Пример:

```
//--- example for CList::DetachCurrent()
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.DetachCurrent();
    if(object==NULL)
    {
        printf("Detach error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- delete element
    delete object;
    //--- delete list
    delete list;
}
```

## DeleteCurrent

Удаляет элемент из текущей позиции списка.

```
bool DeleteCurrent()
```

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элемент.

### Примечание

Если включен механизм управления памятью, память удаляемого элемента освобождается.

### Пример:

```
//--- example for CList::DeleteCurrent()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    if(!list.DeleteCurrent())
    {
        printf("Delete error");
        delete list;
        return;
    }
    //--- delete list
    delete list;
}
```

## Delete

Удаляет элемент из указанной позиции списка.

```
bool Delete(
    int pos        // позиция
)
```

### Параметры

*pos*

[in] позиция удаляемого элемента в списке.

### Возвращаемое значение

true в случае успеха, false - если нет возможности удалить элемент.

### Примечание

Если включен механизм управления памятью, память удаляемого элемента освобождается.

### Пример:

```
---- example for CList::Delete(int)
#include <Arrays\List.mqh>
----
void OnStart()
{
    CList *list=new CList;
    /**
     * if(list==NULL)
     * {
     *     printf("Object create error");
     *     return;
     * }
     * add list elements
     * . . .
     * if(!list.Delete(0))
     * {
     *     printf("Delete error");
     *     delete list;
     *     return;
     * }
     * delete list
     */
}
```

## Clear

Удаляет все элементы списка.

```
void Clear()
```

### Примечание

Если включен механизм управления памятью, память удаляемых элементов освобождается.

### Пример:

```
//--- example for CList::Clear()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    //--- clear list
    list.Clear();
    //--- delete list
    delete list;
}
```

## IndexOf

Получает индекс указанного элемента списка.

```
int IndexOf(
    CObject* element      // указатель на элемент
)
```

### Параметры

*element*  
 [in] указатель на элемент списка.

### Возвращаемое значение

Индекс элемента списка, либо -1.

### Пример:

```
//--- example for CList::IndexOf(CObject*)
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---

    if(list==NULL)
    {
        printf("Object create error");
        return;
    }

    CObject *object=new CObject;
    if(object==NULL)
    {
        printf("Element create error");
        delete list;
        return;
    }

    if(list.Add(object))
    {
        int pos=list.IndexOf(object);
    }

    //--- delete list
    delete list;
}
```

## GetNodeAtIndex

Получает элемент из указанной позиции списка.

```
CObject* GetNodeAtIndex(  
    int pos // позиция  
)
```

### Параметры

*pos*  
[in] позиция элемента в списке.

### Возвращаемое значение

указатель на элемент в случае успеха, NULL - если нет возможности получить указатель.

### Пример:

```
//--- example for CList::GetNodeAtIndex(int)  
#include <Arrays\List.mqh>  
//---  
void OnStart()  
{  
    CList *list=new CList;  
    //---  
    if(list==NULL)  
    {  
        printf("Object create error");  
        return;  
    }  
    //--- add list elements  
    //--- . . .  
    CObject *object=list.GetNodeAtIndex(10);  
    if(object==NULL)  
    {  
        printf("Get node error");  
        delete list;  
        return;  
    }  
    //--- use element  
    //--- . . .  
    //--- do not delete element  
    //--- delete list  
    delete list;  
}
```

## GetFirstNode

Получает первый элемент списка.

```
CObject* GetFirstNode()
```

### Возвращаемое значение

Указатель на первый элемент в случае успеха, NULL - если нет возможности получить указатель.

### Пример:

```
//--- example for CList::GetFirstNode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.GetFirstNode();
    if(object==NULL)
    {
        printf("Get node error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- do not delete element
    //--- delete list
    delete list;
}
```

## GetPrevNode

Получает предыдущий элемент списка.

```
CObject* GetPrevNode()
```

### Возвращаемое значение

Указатель на предыдущий элемент в случае успеха, NULL - если нет возможности получить указатель.

### Пример:

```
//--- example for CList::GetPrevNode()
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.GetPrevNode();
    if(object==NULL)
    {
        printf("Get node error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- do not delete element
    //--- delete list
    delete list;
}
```

## GetCurrentNode

Получает текущий элемент списка.

```
CObject* GetCurrentNode()
```

### Возвращаемое значение

Указатель на текущий элемент в случае успеха, NULL - если нет возможности получить указатель.

### Пример:

```
//--- example for CList::GetCurrentNode()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.GetCurrentNode();
    if(object==NULL)
    {
        printf("Get node error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- do not delete element
    //--- delete list
    delete list;
}
```

## GetNextNode

Получает следующий элемент списка.

```
CObject* GetNextNode()
```

### Возвращаемое значение

Указатель на следующий элемент в случае успеха, NULL - если нет возможности получить указатель.

### Пример:

```
//--- example for CList::GetNextNode()
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.GetNextNode();
    if(object==NULL)
    {
        printf("Get node error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- do not delete element
    //--- delete list
    delete list;
}
```

## GetLastNode

Получает последний элемент списка.

```
CObject* GetLastNode()
```

### Возвращаемое значение

Указатель на последний элемент в случае успеха, NULL - если нет возможности получить указатель.

### Пример:

```
//--- example for CList::GetLastNode()
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add list elements
    //--- . . .
    CObject *object=list.GetLastNode();
    if(object==NULL)
    {
        printf("Get node error");
        delete list;
        return;
    }
    //--- use element
    //--- . . .
    //--- do not delete element
    //--- delete list
    delete list;
}
```

## Sort

Сортирует список.

```
void Sort(
    int mode      // вариант сортировки
)
```

### Параметры

*mode*

[in] Вариант сортировки.

### Возвращаемое значение

Нет.

### Примечание

Сортировка списка производится всегда по возрастанию.

### Пример:

```
---- example for CList::Sort(int)
#include <Arrays\List.mqh>
----
void OnStart()
{
    CList *list=new CList;
    /**
     * if(list==NULL)
     * {
     *     printf("Object create error");
     *     return;
     * }
     * sorting by mode 0
    list.Sort(0);
    /**
     * use list
     * ...
     * delete list
    delete list;
}
```

## MoveToIndex

Перемещает текущий элемент списка в указанную позицию.

```
bool MoveToIndex(
    int pos        // позиция
)
```

### Параметры

*pos*  
[in] позиция в списке для перемещения.

### Возвращаемое значение

true в случае успеха, false - если нет возможности переместить элемент.

### Пример:

```
//--- example for CList::MoveToIndex(int)
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- move current node to begin
    list.MoveToIndex(0);
    //--- use list
    //--- . . .
    //--- delete list
    delete list;
}
```

## Exchange

Меняет элементы списка местами.

```
bool Exchange(
    CObject* node1,      // элемент списка
    CObject* node2       // элемент списка
)
```

### Параметры

*node1*  
 [in] элемент списка

*node2*  
 [in] элемент списка

### Возвращаемое значение

true в случае успеха, false - если нет возможности поменять элементы местами.

### Пример:

```
//--- example for CList::Exchange(CObject*,CObject*)
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- exchange
    list.Exchange(list.GetFirstNode(),list.GetLastNode());
    //--- use list
    //--- . . .
    //--- delete list
    delete list;
}
```

## CompareList

Сравнивает список с другим списком.

```
bool CompareList(
    CList* list        // с кем сравниваем
)
```

### Параметры

*list*

[in] указатель на экземпляр класса CList-источник элементов для сравнения.

### Возвращаемое значение

true в случае если списки равны, false - если нет.

### Пример:

```
//--- example for CList::CompareList(const CList*)
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- create source list
    CList *src=new CList;
    if(src==NULL)
    {
        printf("Object create error");
        delete list;
        return;
    }
    //--- fill source list
    //--- . . .
    //--- compare with another list
    bool result=list.CompareList(src);
    //--- delete lists
    delete src;
    delete list;
}
```

## Search

Ищет элемент равный образцу, в сортированном списке.

```
CObject* Search(
    CObject* element      // образец
)
```

### Параметры

*element*

[in] образец элемента для поиска в списке.

### Возвращаемое значение

Указатель на найденный элемент в случае успеха, NULL - если элемент не найден.

### Пример:

```
//--- example for CList::Search(CObject*)
#include <Arrays\List.mqh>
//---

void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add lists elements
    //--- . . .
    //--- sort list
    list.Sort(0);
    //--- create sample
    CObject *sample=new CObject;
    if(sample==NULL)
    {
        printf("Sample create error");
        delete list;
        return;
    }
    //--- set sample attributes
    //--- . . .
    //--- search element
    if(list.Search(sample)!=NULL) printf("Element found");
    else                         printf("Element not found");
    //--- delete list
```

```
    delete list;  
}
```

## Save

Сохраняет данные списка в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл бинарного файла, открытого ранее при помощи функции FileOpen(...).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CList::Save(int)
#include <Arrays\List.mqh>
//---
void OnStart()
{
    int file_handle;
    CList *list=new CList;
    //---
    if(list!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- add lists elements
    //--- . . .
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!list.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            delete list;
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
}
```

```
//--- delete list  
delete list;  
}
```

## Load

Загружает данные списка из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл бинарного файла, открытого ранее при помощи функции FileOpen(...).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Примечание

При чтении из файла элементов списка, для создания каждого элемента вызывается метод [CList::CreateElement\(\)](#).

### Пример:

```
//--- example for CLoad::Load(int)
#include <Arrays\List.mqh>
//---

void OnStart()
{
    int file_handle;
    CList *list=new CList;
    //---
    if(list!=NULL)
    {
        printf("Object create error");
        return;
    }
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!list.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            delete list;
            FileClose(file_handle);
            //---
            return;
        }
    }
}
```

```
    FileClose(file_handle);
}
//--- use list elements
//--- . . .
//--- delete list
delete list;
}
```

## Type

Получает идентификатор типа списка.

```
virtual int Type()
```

### Возвращаемое значение

Идентификатор типа списка (для CList - 7779).

### Пример:

```
//--- example for CList::Type()
#include <Arrays\List.mqh>
//---
void OnStart()
{
    CList *list=new CList;
    //---
    if(list==NULL)
    {
        printf("Object create error");
        return;
    }
    //--- get list type
    int type=list.Type();
    //--- delete list
    delete list;
}
```

## CTreeNode

Класс CTreeNode является классом узла двоичного дерева CTree.

### Описание

Класс CTreeNode обеспечивает возможность работы с узлами двоичного дерева [CTree](#). В классе реализованы возможности навигации по дереву. Кроме того, реализованы методы работы с файлом.

### Декларация

```
class CArrayObj : public CObject
```

### Заголовок

```
#include <Arrays\TreeNode.mqh>
```

### Иерархия наследования

[CObject](#)

CTreeNode

### Прямые потомки

[CTree](#)

### Методы класса по группам

Атрибуты	
<a href="#">Owner</a>	Получает/устанавливает указатель владельца
<a href="#">Left</a>	Получает/устанавливает указатель левого узла
<a href="#">Right</a>	Получает/устанавливает указатель правого узла
<a href="#">Balance</a>	Получает баланс узла
<a href="#">BalanceL</a>	Получает баланс левой подветви узла
<a href="#">BalanceR</a>	Получает баланс правой подветви узла
<b>Создание нового элемента</b>	
<a href="#">CreateSample</a>	Создает новый экземпляр узла
<b>Сравнение</b>	
<a href="#">RefreshBalance</a>	Пересчитывает баланс узла
<b>Поиск</b>	

<a href="#">GetNext</a>	Получает указатель следующего узла
<b>Ввод/вывод</b>	
<a href="#">SaveNode</a>	Сохраняет данные узла в файл
<a href="#">LoadNode</a>	Загружает данные узла из файла
<b>virtual Type</b>	Получает идентификатор типа узла

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Compare](#)

Практическое применение имеют деревья потомков класса CTreeNode.

Потомок класса CTreeNode должен иметь переопределенные методы: [CreateSample](#), создающий новый экземпляр класса-потомка CTreeNode, [Compare](#), сравнивающий значения ключевых полей класса-потомка CTreeNode, [Type](#) (при необходимости идентификации узла), [SaveNode](#) и [LoadNode](#) (при необходимости работы с файлом).

Рассмотрим пример класса-потомка CTree.

```

//+-----+
//|                                         MyTreeNode.mq5 |
//|                                         Copyright 2010, MetaQuotes Software Corp. |
//|                                         https://www.metaquotes.net/ |
//+-----+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
//---
#include <Arrays\TreeNode.mqh>
//+-----+
//| Описываем класс CMyTreeNode производный от CTreeNode.          |
//+-----+
//| Класс CMyTreeNode.                                              |
//| Назначение: Класс элемента двоичного дерева.                  |
//| Потомок класса CTreeNode.                                         |
//+-----+
class CMyTreeNode : public CTreeNode
{
protected:
    //--- данные пользователя
    long           m_long;           // ключевое поле типа long
    double         m_double;        // переменная пользователя типа double
    string         m_string;        // переменная пользователя типа string
    datetime       m_datetime;      // переменная пользователя типа datetime

public:
    CMyTreeNode();
    //--- методы доступа к данным пользователя
    long           GetLong(void)           { return(m_long); }

```

```

void SetLong(long value) { m_long=value; }
double GetDouble(void) { return(m_double); }
void SetDouble(double value) { m_double=value; }
string GetString(void) { return(m_string); }
void SetString(string value) { m_string=value; }
datetime GetDateTime(void) { return(m_datetime); }
void SetDateTime(datetime value) { m_datetime=value; }

//--- методы работы с файлами
virtual bool Save(int file_handle);
virtual bool Load(int file_handle);

protected:
virtual int Compare(const CObject *node,int mode);
//--- метод создания экземпляра класса
virtual CTreeNode* CreateSample();
};

//+-----+
//| Конструктор класса CMyTreeNode.
//| INPUT: нет.
//| OUTPUT: нет.
//| REMARK: нет.
//+-----+
void CMyTreeNode::CMyTreeNode()
{
//--- инициализация данных пользователя
m_long =0;
m_double =0.0;
m_string ="";
m_datetime =0;
}

//+-----+
//| Сравнение с другим узлом дерева по указанному алгоритму.
//| INPUT: node - элемент массива для сравнения,
//|        mode - идентификатор алгоритма сравнения.
//| OUTPUT: результат сравнения (>0,0,<0).
//| REMARK: нет.
//+-----+
int CMyTreeNode::Compare(const CObject *node,int mode)
{
//--- параметр mode игнорируется, т.к. алгоритм построения дерева единственный
int res=0;
//--- явное преобразование типа
CMyTreeNode *n=node;
res=(int) (m_long-n.m_long);
//---
return(res);
}

//+-----+
//| Создание нового экземпляра класса.
//| INPUT: нет.

```

```

//| OUTPUT: указатель на новый экземпляр класса CMyTreeNode.
//| REMARK: нет.
//+-----+
CTreeNode* CMyTreeNode::CreateSample()
{
    CMyTreeNode *result=new CMyTreeNode;
//---
    return(result);
}
//+-----+
//| Запись данных узла дерева в файл.
//| INPUT: file_handle -хендл ранее открытого для записи файла.
//| OUTPUT: true если OK, иначе false.
//| REMARK: нет.
//+-----+
bool CMyTreeNode::Save(int file_handle)
{
    uint i=0,len;
//--- проверки
    if(file_handle<0) return(false);
//--- собственно запись данных пользователя
//--- запись переменной пользователя типа long
    if(FileWriteLong(file_handle,m_long)!=sizeof(long)) return(false);
//--- запись переменной пользователя типа double
    if(FileWriteDouble(file_handle,m_double)!=sizeof(double)) return(false);
//--- запись переменной пользователя типа string
    len=StringLen(m_string);
//--- запись длины строки
    if(FileWriteInteger(file_handle,len,INT_VALUE)!=INT_VALUE) return(false);
//--- запись собственно строки
    if(len!=0 && FileWriteString(file_handle,m_string,len)!=len) return(false);
//--- запись переменной пользователя типа datetime
    if(FileWriteLong(file_handle,m_datetime)!=sizeof(long)) return(false);
//---
    return(true);
}
//+-----+
//| Чтение данных узла дерева из файла.
//| INPUT: file_handle -хендл ранее открытого для чтения файла.
//| OUTPUT: true если OK, иначе false.
//| REMARK: нет.
//+-----+
bool CMyTreeNode::Load(int file_handle)
{
    uint i=0,len;
//--- проверки
    if(file_handle<0) return(false);
//--- собственно чтение
    if(FileIsEnding(file_handle)) return(false);

```

```
//--- чтение переменной пользователя типа char
//--- чтение переменной пользователя типа long
    m_long=FileReadLong(file_handle);
//--- чтение переменной пользователя типа double
    m_double=FileReadDouble(file_handle);
//--- чтение переменной пользователя типа string
//--- чтение длины строки
    len=FileReadInteger(file_handle, INT_VALUE);
//--- чтение собственно строки
    if(len!=0) m_string=FileReadString(file_handle, len);
    else      m_string="";
//--- чтение переменной пользователя типа datetime
    m_datetime=FileReadLong(file_handle);
//---
    return(true);
}
```

## Owner

Получает указатель узла-владельца.

```
CTreeNode* Owner()
```

### Возвращаемое значение

Указатель узла-владельца.

## Owner

Устанавливает указатель узла-владельца.

```
void Owner(  
    CTreeNode* node      // узел  
)
```

### Параметры

*node*

[in] Новое значение указателя узла-владельца.

### Возвращаемое значение

Нет.

## Left

Получает указатель левого узла.

```
CTreeNode* Left()
```

### Возвращаемое значение

Указатель левого узла.

## Left

Устанавливает указатель левого узла.

```
void Left(
    CTreeNode* node      // узел
)
```

### Параметры

*node*

[in] Новое значение указателя левого узла.

### Возвращаемое значение

Нет.

## Right

Получает указатель правого узла.

```
CTreeNode* Right()
```

### Возвращаемое значение

Указатель правого узла.

## Right

Устанавливает указатель правого узла.

```
void Right(
    CTreeNode* node      // узел
)
```

### Параметры

*node*

[in] Новое значение указателя правого узла.

### Возвращаемое значение

Нет.

## Balance

Получает баланс узла.

```
int Balance() const
```

### Возвращаемое значение

Баланс узла.

## BalanceL

Получает баланс левой подветви узла.

```
int BalanceL() const
```

### Возвращаемое значение

Баланс левой подветви узла.

## BalanceR

Получает баланс правой подветви узла.

```
int BalanceR() const
```

### Возвращаемое значение

Баланс правой подветви узла.

## CreateSample

Создает новый экземпляр узла.

```
virtual CTreeNode* CreateSample()
```

### Возвращаемое значение

Указатель нового экземпляра узла, либо NULL.

## RefreshBalance

Пересчитывает баланс узла.

```
int RefreshBalance()
```

### Возвращаемое значение

Баланс узла.

## GetNext

Получает указатель следующего узла.

```
CTreeNode* GetNext (
    CTreeNode* node           // узел
)
```

### Параметры

*node*

[in] Узел начала поиска.

### Возвращаемое значение

Указатель следующего узла.

## SaveNode

Записывает данные узла в файл.

```
bool SaveNode(
    int file_handle // хэндл
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого для записи бинарного файла.

### Возвращаемое значение

true в случае удачи, иначе false.

## LoadNode

Читает данные узла из файла.

```
bool LoadNode(
    int      file_handle,      // хэндл
    CTreeNode* main           // узел
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого для чтения бинарного файла.

*main*

[in] Узел для данных.

### Возвращаемое значение

true в случае удачи, иначе false.

## Type

Получает идентификатор типа узла.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа узла.

## CTree

Класс CTree является классом двоичного дерева экземпляров класса CTreeNode и его наследников.

### Описание

Класс CTree обеспечивает возможность работы с двоичным деревом экземпляров класса [CTreeNode](#) и его наследников. В классе реализованы возможности добавления/вставки/удаления элементов дерева, поиска в дереве. Кроме того, реализованы методы работы с файлом.

Следует отметить что, в классе CTree не реализован механизм управления динамической памятью (в отличии от классов [CList](#) и [CArrayObj](#)). Все узлы дерева удаляются с освобождением памяти.

### Декларация

```
class CTree : public CTreeNode
```

### Заголовок

```
#include <Arrays\Tree.mqh>
```

### Иерархия наследования

```
CObject  
CTreeNode  
CTree
```

### Методы класса по группам

Атрибуты	
<a href="#">Root</a>	Получает корневой узел дерева
<b>Создание нового элемента</b>	
<a href="#">virtual CreateElement</a>	Создает новый экземпляр узла
<b>Наполнение</b>	
<a href="#">Insert</a>	Добавляет узел в дерево
<b>Удаление</b>	
<a href="#">Detach</a>	Изымает указанный узел из дерева
<a href="#">Delete</a>	Удаляет указанный узел из дерева
<a href="#">Clear</a>	Удаляет все узлы дерева
<b>Поиск</b>	
<a href="#">Find</a>	Ищет в дереве узел по образцу
<b>Ввод/вывод</b>	

virtual <a href="#">Save</a>	Сохраняет данные дерева в файл
virtual <a href="#">Load</a>	Загружает данные дерева из файла
virtual <a href="#">Type</a>	Получает идентификатор типа дерева

**Методы унаследованные от CObject**Prev, Prev, Next, Next, [Compare](#)**Методы унаследованные от CTreeNode**

Parent, Parent, [Left](#), [Left](#), [Right](#), [Right](#), [Balance](#), [BalanceL](#), [BalanceR](#), [RefreshBalance](#), [GetNext](#), [SaveNode](#), [LoadNode](#)

Практическое применение имеют деревья потомков класса CTreeNode - потомки класса CTree.

Потомок класса CTree должен иметь переопределенный метод [CreateElement](#), создающий новый экземпляр класса-потомка [CTreeNode](#).

Рассмотрим пример класса-потомка CTree.

```

//+-----+
//|                                                 MyTree.mq5 |
//|                                                 Copyright 2010, MetaQuotes Software Corp. |
//|                                                 https://www.metaquotes.net/ |
//+-----+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
//---
#include <Arrays\Tree.mqh>
#include "MyTreeNode.mqh"
//---
input int extCountedNodes = 100;
//+-----+
//| Описываем класс CMyTree производный от CTree.          |
//+-----+
//| Класс CMyTree.                                         |
//| Назначение: Построение и навигация двоичного дерева поиска. |
//+-----+
class CMyTree : public CTree
{
public:
    //--- методы поиска в дереве по данным пользователя
    CMyTreeNode*      FindByLong(long find_long);
    //--- метод создания элемента дерева
    virtual CTreeNode *CreateElement();
};

//---
CMyTree MyTree;
//+-----+
//| Создание нового узла дерева.                           |

```

```

//| INPUT: нет.
//| OUTPUT: указатель на новый узел дерева если OK, либо NULL.
//| REMARK: нет.
//+-----+
CTreeNode *CMyTree::CreateElement()
{
    CMyTreeNode *node=new CMyTreeNode;
//---
    return(node);
}
//+-----+
//| Поиск элемента в списке по значению m_long.
//| INPUT: find_long - искомое значение.
//| OUTPUT: указатель найденного элемента списка, либо NULL.
//| REMARK: нет.
//+-----+
CMyTreeNode* CMyTree::FindByLong(long find_long)
{
    CMyTreeNode *res=NULL;
    CMyTreeNode *node;
//--- создаем узел дерева для передачи параметра поиска
    node=new CMyTreeNode;
    if(node==NULL) return(NULL);
    node.SetLong(find_long);
//---
    res=Find(node);
    delete node;
//---
    return(res);
}
//+-----+
//| скрипт "тестирование класса CMyTree"
//+-----+
//--- массив для инициализации строк
string str_array[11]={"p","oo","iii","uuuu","yyyyy","ttttt","rrrrr","eee","ww","q","999"};
//---
int OnStart() export
{
    int          i;
    uint         pos;
    int          beg_time,end_time;
    CMyTreeNode *node; //--- временный указатель на экземпляр класса CMyTreeNode
//---
    printf("Start test %s.",__FILE__);
//--- Наполняем MyTree экземплярами класса MyTreeNode в количестве extCountedNodes.
    beg_time=GetTickCount();
    for(i=0;i<extCountedNodes;i++)
    {
        node=MyTree.CreateElement();

```

```

if(node==NULL)
{
    //--- аварийный выход
    printf("%s (%4d): create error", __FILE__, __LINE__);
    return(__LINE__);
}

NodeSetData(node,i);
node.SetLong(i);
MyTree.Insert(node);
}

end_time=GetTickCount();
printf("Время заполнения MyTree %d мс.",end_time-beg_time);
//--- Создаем временное дерево TmpMyTree.

CMyTree TmpMyTree;
//--- Изыаем 50% элементов из дерева (все четные)
//--- и добавляем их во временное дерево TmpMyTree.

beg_time=GetTickCount();
for(i=0;i<extCountedNodes;i+=2)
{
    node=MyTree.FindByLong(i);
    if(node!=NULL)
        if(MyTree.Detach(node)) TmpMyTree.Insert(node);
}

end_time=GetTickCount();
printf("Время удаления %d элементов из MyTree %d мс.",extCountedNodes/2,end_time-be
//--- Возвращаем изятое обратно

node=TmpMyTree.Root();
while(node!=NULL)
{
    if(TmpMyTree.Detach(node)) MyTree.Insert(node);
    node=TmpMyTree.Root();
}

//--- Проверяем работу метода Save(int file_handle);

int file_handle;
file_handle=FileOpen("MyTree.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
if(file_handle>=0)
{
    if(!MyTree.Save(file_handle))
    {
        //--- ошибка записи в файл
        //--- аварийный выход
        printf("%s: Error %d in %d!",__FILE__,GetLastError(),__LINE__);
        //--- уходя, закройте файл !!!
        FileClose(file_handle);
        return(__LINE__);
    }
    FileClose(file_handle);
}

//--- Проверяем работу метода Load(int file_handle);

```

```

file_handle=FileOpen("MyTree.bin",FILE_READ|FILE_BIN|FILE_ANSI);
if(file_handle>=0)
{
    if(!TmpMyTree.Load(file_handle))
    {
        //--- ошибка чтения из файла
        //--- аварийный выход
        printf("%s: Error %d in %d!",__FILE__,GetLastError(),__LINE__);
        //--- уходя, закройте файл !!!
        FileClose(file_handle);
        return(__LINE__);
    }
    FileClose(file_handle);
}
//---

MyTree.Clear();
TmpMyTree.Clear();
//---

printf("End test %s. OK!",__FILE__);
//---

return(0);
}

//-----+
//| Функция вывода содержимого node в журнал
//-----+
void NodeToLog(CMyTreeNode *node)
{
    printf("%I64d,%f,'%s','%s',
           node.GetLong(),node.GetDouble(),
           node.GetString(),TimeToString(node.GetDateTime()));
}

//-----+
//| Функция "заполнения" node случайными значениями
//-----+
void NodeSetData(CMyTreeNode *node,int mode)
{
    if(mode%2==0)
    {
        node.SetLong(mode*MathRand());
        node.SetDouble(MathPow(2.02,mode)*MathRand());
    }
    else
    {
        node.SetLong(mode*(long)(-1)*MathRand());
        node.SetDouble(-MathPow(2.02,mode)*MathRand());
    }
    node.SetString(str_array[mode%10]);
    node.SetDateTime(10000*mode);
}

```



## Root

Получает корневой узел дерева.

```
CTreeNode* Root() const
```

### Возвращаемое значение

Указатель корневого узла дерева.

## CreateElement

Создает новый экземпляр узла.

```
virtual CTreeNode* CreateElement()
```

### Возвращаемое значение

Указатель нового экземпляра узла, либо NULL.

## Insert

Добавляет узел в дерево.

```
CTreeNode* Insert(
    CTreeNode* new_node           // узел
)
```

### Параметры

*new\_node*

[in] Указатель узла для вставки в дерево.

### Возвращаемое значение

Указатель узла-владельца, либо NULL.

## Detach

Изымаает указанный узел из дерева.

```
bool Detach(  
    CTreeNode* node      // узел  
)
```

### Параметры

*node*

[in] Указатель узла для изъятия.

### Возвращаемое значение

true-в случае удачи, иначе false.

### Примечание

После изъятия из дерева указатель узла не освобождается. Дерево балансируется.

## Delete

Удаляет указанный узел из дерева.

```
bool Delete(
    CTreeNode* node      // узел
)
```

### Параметры

*node*

[in] Указатель узла для удаления.

### Возвращаемое значение

true-в случае удачи, иначе false.

### Примечание

После удаления из дерева указатель узла освобождается. Дерево балансируется.

## Clear

Удаляет все узлы дерева.

```
void Clear()
```

### Возвращаемое значение

Нет.

### Примечание

После удаления из дерева указатели узлов освобождаются.

## Find

Ищет в дереве узел по образцу.

```
CTreeNode* Find(
    CTreeNode* node      // узел
)
```

### Параметры

*node*

[in] Узел, содержащий данные-образец поиска.

### Возвращаемое значение

Указатель найденного узла, либо NULL.

## Save

Записывает данные дерева в файл.

```
virtual bool Save(
    int file_handle          // хэндл
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого для записи бинарного файла.

### Возвращаемое значение

true в случае удачи, иначе false.

## Load

Читает данные дерева из файла.

```
virtual bool Load(
    int file_handle // хэндл
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого для чтения бинарного файла.

### Возвращаемое значение

true в случае удачи, иначе false.

## Type

Получает идентификатор типа дерева.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа дерева.

## Шаблонные коллекции данных

Библиотека содержит классы и интерфейсы для определения шаблонных коллекций, которые, в свою очередь, дают пользователю возможность создавать строго типизированные коллекции. Они обеспечивают большее удобство и высокую производительность работы с данными, чем обычные типизированные коллекции.

Библиотека размещается в рабочем каталоге терминала в папке `Include\Generic`.

Объекты:

Объект	Описание	Тип
<a href="#">ICollection</a>	Интерфейс для реализации шаблонных коллекций данных	INTERFACE
<a href="#">IEqualityComparer</a>	Интерфейс для реализации объектов, которые можно сравнивать между собой	INTERFACE
<a href="#">IComparable</a>	Интерфейс для реализации объектов, которые можно сравнивать между собой посредством отношения "больше, меньше или равно"	INTERFACE
<a href="#">IComparer</a>	Интерфейс для реализации универсального класса, который сравнивает два объекта типа Т на отношение "больше, меньше или равно"	INTERFACE
<a href="#">IEqualityComparer</a>	Интерфейс для реализации универсального класса, который сравнивает два объекта типа Т на равенство	INTERFACE
<a href="#"> IList</a>	Интерфейс для реализации шаблонных списков данных	INTERFACE
<a href="#">IMap</a>	Интерфейс для реализации шаблонных коллекций пар "ключ – значение"	INTERFACE
<a href="#">ISet</a>	Интерфейс для реализации шаблонных множеств данных	INTERFACE
<a href="#">CDefaultComparer</a>	Вспомогательный класс, реализующий шаблонный интерфейс <code>IComparer&lt;T&gt;</code> на основе глобальных методов <code>Compare</code>	CLASS

<a href="#">CDefaultEqualityComparer</a>	Вспомогательный класс, реализующий шаблонный интерфейс <code>IEqualityComparer&lt;T&gt;</code> глобальными методами <code>Equals&lt;T&gt;</code> и <code>GetHashCode</code>	CLASS
<a href="#">CArrayList</a>	Шаблонный класс, реализующий интерфейс <code>IList&lt;T&gt;</code>	CLASS
<a href="#">CKeyValuePair</a>	Класс реализует пару "ключ – значение"	CLASS
<a href="#">CHashMap</a>	Шаблонный класс, реализующий интерфейс <code>IMap&lt; TKey, TValue &gt;</code>	CLASS
<a href="#">CHashSet</a>	Шаблонный класс, реализующий интерфейс <code>ISet&lt;T&gt;</code>	CLASS
<a href="#">CLinkedListNode</a>	Вспомогательный класс, необходимый для реализации класса <code>CLinkedListNode&lt;T&gt;</code>	CLASS
<a href="#">CLinkedList</a>	Шаблонный класс, реализующий интерфейс <code>ICollection&lt;T&gt;</code>	CLASS
<a href="#">CQueue</a>	Шаблонный класс, реализующий интерфейс <code>ICollection&lt;T&gt;</code>	CLASS
<a href="#">CRedBlackTreeNode</a>	Вспомогательный класс, необходимый для реализации класса <code>CRedBlackTree&lt;T&gt;</code>	CLASS
<a href="#">CRedBlackTree</a>	Шаблонный класс, реализующий интерфейс <code>ICollection&lt;T&gt;</code>	CLASS
<a href="#">CSortedMap</a>	Шаблонный класс, реализующий интерфейс <code>IMap&lt; TKey, TValue &gt;</code>	CLASS
<a href="#">CSortedSet</a>	Шаблонный класс, реализующий интерфейс <code>ISet&lt;T&gt;</code>	CLASS
<a href="#">CStack</a>	Шаблонный класс, реализующий интерфейс <code>ICollection&lt;T&gt;</code>	CLASS

Глобальные методы:

Метод	Описание
<a href="#"><u>ArrayBinarySearch</u></a>	Ищет указанное значение в отсортированном по возрастанию одномерном массиве, используя интерфейс <code>IComparable&lt;T&gt;</code> для сравнения элементов
<a href="#"><u>ArrayIndexOf</u></a>	Ищет первое вхождение значения в одномерном массиве
<a href="#"><u>ArrayLastIndexOf</u></a>	Ищет последнее вхождение значения в одномерном массиве
<a href="#"><u>ArrayReverse</u></a>	Изменяет последовательность элементов в одномерном массиве
<a href="#"><u>Compare</u></a>	Сравнивает два значения на отношение "Больше, меньше или равно"
<a href="#"><u>Equals</u></a>	Сравнивает два значения на равенство
<a href="#"><u>GetHashCode</u></a>	Вычисляет значение хэш-кода

## ICollection<T>

ICollection<T> – интерфейс для реализации шаблонных коллекций данных.

### Описание

Интерфейс ICollection<T> определяет основные методы для работы с коллекциями: подсчет количества элементов, очистку коллекции, добавление и удаление элементов и другие.

### Декларация

```
template<typename T>
interface ICollection
```

### Заголовок

```
#include <Generic\Interfaces\ICollection.mqh>
```

### Иерархия наследования

ICollection

#### Прямые потомки

[CLinkedList](#), [CQueue](#), [CRedBlackTree](#), [CStack](#), [IList](#), [IMap](#), [ISet](#)

### Методы класса

Метод	Описание
<a href="#">Add</a>	Добавляет элемент в коллекцию
<a href="#">Count</a>	Возвращает количество элементов в коллекции
<a href="#">Contains</a>	Определяет, содержит ли коллекция элемент с указанным значением
<a href="#">CopyTo</a>	Копирует все элементы коллекции в указанный массив, начиная с определенного индекса
<a href="#">Clear</a>	Удаляет все элементы коллекции
<a href="#">Remove</a>	Удаляет первое вхождение указанного элемента из коллекции

## Add

Добавляет элемент в коллекцию.

```
bool Add(  
    T   value      // значение элемента  
);
```

### Параметры

*value*

[in] Значение элемента для добавления.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Count

Возвращает количество элементов в коллекции.

```
int Count();
```

### Возвращаемое значение

Возвращает количество элементов.

## Contains

Определяет, содержит ли коллекция элемент с указанным значением.

```
bool Contains(  
    T   item      // искомое значение  
);
```

### Параметры

*item*

[in] Искомое значение.

### Возвращаемое значение

Возвращает true, если в коллекции есть элемент с указанным значением, иначе false.

## CopyTo

Копирует все элементы коллекции в указанный массив, начиная с определенного индекса.

```
int CopyTo(  
    T&          dst_array[],      // массив для записи  
    const int   dst_start=0        // начальный индекс для записи  
);
```

### Параметры

`&dst_array[]`

[out] Массив, в который будут записаны элементы коллекции.

`dst_start=0`

[in] Индекс в массиве, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных элементов.

## Clear

Удаляет все элементы коллекции.

```
void Clear();
```

## Remove

Удаляет первое вхождение указанного элемента из коллекции.

```
bool Remove(  
    T    item      // значение элемента  
);
```

### Параметры

*item*

[in] Значение элемента, которое нужно удалить.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## IEqualityComparable<T>

Интерфейс `IEqualityComparable<T>` – интерфейс для реализации объектов, которые можно сравнивать между собой.

### Описание

Интерфейс `IEqualityComparable<T>` определяет методы получения хэш-кода текущего объекта и его сравнения на равенство с другим объектом того же типа.

### Декларация

```
template<typename T>
interface IEqualityComparable
```

### Заголовок

```
#include <Generic\Interfaces\IEqualityComparable.mqh>
```

### Иерархия наследования

`IEqualityComparable`

#### Прямые потомки

[IComparable](#)

### Методы класса

Метод	Описание
<a href="#">Equals</a>	Сравнивает текущий объект с заданным значением
<a href="#">HashCode</a>	Вычисляет значение хэш-кода для текущего объекта

## Equals

Сравнивает текущий объект с заданным значением.

```
bool Equals(  
    T   value      // значение для сравнения  
);
```

### Параметры

*value*

[in] Значение, с которым сравнивается текущий объект.

### Возвращаемое значение

Возвращает true, если объекты равны, иначе false.

## HashCode

Вычисляет значение хэш-кода для текущего объекта.

```
int HashCode();
```

### Возвращаемое значение

Возвращает хэш-код.

## IComparable<T>

Интерфейс `IComparable<T>` – интерфейс для реализации объектов, которые можно сравнивать между собой посредством отношения "больше, меньше или равно".

### Описание

Интерфейс `IComparable<T>` определяет метод сравнения текущего объекта с другим того же типа, на основе которого коллекция этих объектов может быть отсортирована.

### Декларация

```
template<typename T>
interface IComparable : public IEqualityComparable<T>
```

### Заголовок

```
#include <Generic\Interfaces\IComparable.mqh>
```

### Иерархия наследования

[IEqualityComparable](#)

`IComparable`

### Прямые потомки

[CKeyValuePair](#)

### Методы класса

Метод	Описание
<a href="#">Compare</a>	Сравнивает текущий объект с указанным значением

## Compare

Сравнивает текущий объект с указанным значением.

```
int Compare(
    T   value      // значение для сравнения
);
```

### Параметры

*value*

[in] Значение, с которым сравнивается текущий объект.

### Возвращаемое значение

Возвращает число, выражающее отношение текущего и переданного объекта:

- результат меньше нуля — текущий объект меньше переданного
- результат равен нулю — текущий объект равен переданному
- результат больше нуля — текущий объект больше переданного

## IComparer<T>

Интерфейс `IComparer<T>` – интерфейс для реализации универсального класса, который проводит сравнение двух объектов типа `T` между собой посредством отношения "Больше, меньше или равно".

### Описание

Интерфейс `IComparer<T>` определяет метод сравнения двух объектов типа `T`, на основе которого коллекция этих объектов может быть отсортирована.

### Декларация

```
template<typename T>
interface IComparer
```

### Заголовок

```
#include <Generic\Interfaces\IComparer.mqh>
```

### Иерархия наследования

`IComparer`

#### Прямые потомки

[CDefaultComparer](#)

### Методы класса

Метод	Описание
<a href="#">Compare</a>	Сравнивает два значения типа <code>T</code>

## Compare

Сравнивает два значения типа Т.

```
int Compare(
    Т   x,      // первое значение
    Т   y,      // второе значение
);
```

### Параметры

*x*

[in] Первое значение для сравнения.

*y*

[in] Второе значение для сравнения.

### Возвращаемое значение

Возвращает число, выражающее отношение двух сравниемых значений:

- результат меньше нуля – *x* меньше *y* (*x*<*y*)
- результат равен нулю – *x* равен *y* (*x*=*y*)
- результат больше нуля – *x* больше *y* (*x*>*y*)

## IEqualityComparer<T>

Интерфейс `IEqualityComparer<T>` – интерфейс для реализации универсального класса, который проводит сравнение двух объектов типа `T`.

### Описание

Интерфейс `IEqualityComparer<T>` определяет методы получения хэш-кода объекта типа `T` и сравнения на равенство двух объектов типа `T`.

### Декларация

```
template<typename T>
interface IEqualityComparer
```

### Заголовок

```
#include <Generic\Interfaces\IEqualityComparer.mqh>
```

### Иерархия наследования

`IEqualityComparer`

#### Прямые потомки

[CDefaultEqualityComparer](#)

### Методы класса

Метод	Описание
<a href="#">Equals</a>	Сравнивает два значения типа <code>T</code>
<a href="#">HashCode</a>	Вычисляет значение хэш-кода от объекта типа <code>T</code>

## Equals

Сравнивает два значения типа Т.

```
bool Equals(  
    Т   x,      // первое значение  
    Т   y,      // второе значение  
);
```

### Параметры

*x*

[in] Первое значение для сравнения.

*y*

[in] Второе значение для сравнения.

### Возвращаемое значение

Возвращает true, если значения равны, иначе false.

## HashCode

Вычисляет значение хэш-кода от объекта типа Т.

```
int HashCode(  
    T   value      // объект для вычисления  
);
```

### Параметры

*value*

[in] Объект, для которого нужно получить хэш-код.

### Возвращаемое значение

Возвращает хэш-код.

## IList<T>

Интерфейс `IList<T>` – интерфейс для реализации шаблонных списков данных.

### Описание

Интерфейс `IList<T>` определяет основные методы для работы со списками: доступ к элементу по индексу, поиск и удаление элемента, сортировку и другие.

### Декларация

```
template<typename T>
interface IList : public ICollection<T>
```

### Заголовок

```
#include <Generic\Interfaces\IList.mqh>
```

### Иерархия наследования

[ICollection](#)

`IList`

### Прямые потомки

[CArrayList](#)

### Методы класса

Метод	Описание
<a href="#">TryGetValue</a>	Получает элемент из списка по заданному индексу
<a href="#">TrySetValue</a>	Изменяет значение из списка по заданному индексу
<a href="#">Insert</a>	Вставляет элемент в список по указанному индексу
<a href="#">IndexOf</a>	Ищет первое вхождение значения в списке
<a href="#">LastIndexOf</a>	Ищет последнее вхождение значения в списке
<a href="#">RemoveAt</a>	Удаляет элемент из списка по указанному индексу

## TryGetValue

Получает элемент из списка по заданному индексу.

```
bool TryGetValue(  
    const int   index,      // индекс элемента  
    T&         value       // переменная для записи  
);
```

### Параметры

*index*

[in] Индекс элемента из списка.

*&value*

[out] Переменная, в которую будет записано указанное значение элемента из списка.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## TrySetValue

Изменяет значение из списка по заданному индексу.

```
bool TrySetValue(
    const int   index,      // индекс элемента
    Т          value       // новое значение
);
```

### Параметры

*index*

[in] Индекс элемента из списка.

*value*

[in] Новое значение, которое нужно присвоить указанному элементу списка.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Insert

Вставляет элемент в список по указанному индексу.

```
bool Insert(  
    const int   index,      // индекс для вставки  
    T          item        // значение для вставки  
);
```

### Параметры

*index*

[in] Индекс для вставки.

*item*

[in] Значение, которое необходимо вставить по указанному индексу.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## IndexOf

Ищет первое вхождение значения в списке.

```
int IndexOf(  
    T item // искомое значение  
) ;
```

### Параметры

*item*

[in] Искомое значение.

### Возвращаемое значение

Возвращает индекс первого найденного элемента. Если значение не найдено, возвращает -1.

## LastIndexOf

Ищет последнее вхождение значения в списке.

```
int LastIndexOf(
    T   item      // искомое значение
);
```

### Параметры

*item*

[in] Искомое значение.

### Возвращаемое значение

Возвращает индекс последнего найденного элемента. Если значение не найдено, возвращает -1.

## RemoveAt

Удаляет элемент из списка по указанному индексу.

```
bool RemoveAt(  
    const int   index      // индекс элемента  
);
```

### Параметры

*index*

[in] Индекс элемента, который необходимо удалить.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## IMap< TKey, TValue >

Интерфейс IMap< TKey, TValue > – интерфейс для реализации шаблонных коллекций пар "ключ – значение".

### Описание

Интерфейс IMap< TKey, TValue > определяет основные методы для работы с коллекциями, данные в которых хранятся в виде пар "ключ – значение".

### Декларация

```
template<typename TKey, typename TValue>
interface IMap : public ICollection<TKey>
```

### Заголовок

```
#include <Generic\Interfaces\IMap.mqh>
```

### Иерархия наследования

[ICollection](#)

IMap

### Прямые потомки

[CHashMap](#), [CSortedMap](#)

### Методы класса

Метод	Описание
<a href="#">Add</a>	Добавляет пару "ключ – значение" в коллекцию
<a href="#">Contains</a>	Определяет, содержит ли коллекция пару "ключ – значение" с указанным ключом и значением
<a href="#">Remove</a>	Удаляет первое вхождение пары "ключ – значение" с указанным ключом из коллекции
<a href="#">TryGetValue</a>	Получает элемент из коллекции по заданному ключу
<a href="#">TrySetValue</a>	Изменяет значение пары "ключ – значение" из коллекции по заданному ключу
<a href="#">CopyTo</a>	Копирует все пары "ключ – значение" из коллекции в указанные массивы, начиная с определенного индекса

## Add

Добавляет пару "ключ — значение" в коллекцию.

```
bool Add(  
    TKey     key,      //  ключ  
    TValue   value     //  значение  
);
```

### Параметры

*key*

[in] Ключ.

*value*

[in] Значение.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Contains

Определяет, содержит ли коллекция пару "ключ — значение" с указанным ключом и значением.

```
bool Contains(
    TKey     key,      //  ключ
    TValue   value     //  значение
);
```

### Параметры

*key*  
[in] Ключ.

*value*  
[in] Значение.

### Возвращаемое значение

Возвращает true, если в коллекции есть пара "ключ — значение" с указанными ключом и значением, иначе false.

## Remove

Удаляет первое вхождение пары "ключ — значение" с указанным ключом из коллекции.

```
bool Remove(  
    TKey key // Ключ  
) ;
```

### Параметры

*key*

[in] Ключ.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## TryGetValue

Получает элемент из коллекции по заданному ключу.

```
bool TryGetValue(  
    TKey      key,          // Ключ  
    TValue&   value        // Переменная для записи значения  
);
```

### Параметры

*key*

[in] Ключ.

*&value*

[out] Переменная, в которую будет записано указанное значение пары "ключ — значение".

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## TrySetValue

Изменяет значение пары "ключ — значение" из коллекции по заданному ключу.

```
bool TrySetValue(  
    TKey     key,      // Ключ  
    TValue   value     // новое значение  
);
```

### Параметры

*key*

[in] Ключ.

*value*

[in] Новое значение, которое нужно присвоить указанной паре "ключ-значение".

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## CopyTo

Копирует все пары "ключ — значение" из коллекции в указанные массивы, начиная с определенного индекса.

```
int CopyTo(  
    TKey&      dst_keys[],           // массив для записи ключей  
    TValue&    dst_values[],        // массив для записи значений  
    const int   dst_start=0          // начальный индекс для записи  
) ;
```

### Параметры

*&dst\_keys[]*

[out] Массив, в который будут записаны все ключи из коллекции.

*&dst\_values[]*

[out] Массив, в который будут записаны значения соответствующих ключей из коллекции.

*dst\_start=0*

[in] Индекс в массивах, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных пар "ключ — значение".

## ISet<T>

Интерфейс ISet<T> – интерфейс для реализации шаблонных множеств данных.

### Описание

Интерфейс ISet<T> определяет основные методы для работы с множествами: объединение и пересечение множеств, определение строгих и нестрогих подмножеств и другие.

### Декларация

```
template<typename T>
interface ISet : public ICollection<T>
```

### Заголовок

```
#include <Generic\Interfaces\ISet.mqh>
```

### Иерархия наследования

[ICollection](#)

ISet

### Прямые потомки

[CHashSet](#), [CSortedSet](#)

### Методы класса

Метод	Описание
<a href="#">ExceptWith</a>	Выполняет операцию разности текущей и переданной коллекции (массива)
<a href="#">IntersectWith</a>	Выполняет операцию пересечения текущей и переданной коллекции (массива)
<a href="#">SymmetricExceptWith</a>	Выполняет операцию симметричной разности текущей и переданной коллекции (массива)
<a href="#">UnionWith</a>	Выполняет операцию объединения текущей и переданной коллекции (массива)
<a href="#">IsProperSubsetOf</a>	Определяет, является ли текущее множество строгим подмножеством заданной коллекции или массива
<a href="#">IsProperSupersetOf</a>	Определяет, является ли текущее множество строгим надмножеством заданной коллекции или массива
<a href="#">IsSubsetOf</a>	Определяет, является ли текущее множество подмножеством заданной коллекции или массива

<a href="#"><u>IsSupersetOf</u></a>	Определяет, является ли текущее множество надмножеством заданной коллекции или массива
<a href="#"><u>Overlaps</u></a>	Определяет, пересекается ли текущее множество с заданной коллекцией или массивом
<a href="#"><u>SetEquals</u></a>	Определяет, содержит ли текущее множество все элементы из заданной коллекции или массива

## ExceptWith

Выполняет операцию разности текущей и переданной коллекции (массива). То есть удаляет из текущей коллекции (массива) все элементы, которые есть также и в указанной коллекции (массиве).

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void ExceptWith(
    ICollection<T>* collection      // коллекция
);
```

Версия для работы с массивом.

```
void ExceptWith(
    T& array[]                  // массив
);
```

### Параметры

`*collection`

[in] Коллекция, которая будет вычитаться из текущего множества.

`&collection[]`

[in] Массив, который будет вычитаться из текущего множества.

### Примечание

Результат записывается в текущую коллекцию (массив).

## IntersectWith

Выполняет операцию пересечения текущей и переданной коллекции (массива). То есть оставляет в текущей коллекции только те элементы, которые имеются также и в указанной коллекции (массиве).

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void IntersectWith(
    ICollection<T>* collection      // коллекция
);
```

Версия для работы с массивом.

```
void IntersectWith(
    T& array[]                      // массив
);
```

### Параметры

`*collection`

[in] Коллекция, с которой будет строиться произведение.

`&collection[]`

[in] Массив, с которым будет строиться произведение.

### Примечание

Результат записывается в текущую коллекцию (массив).

## SymmetricExceptWith

Выполняет операцию симметричной разности текущей и переданной коллекции (массива). То есть в текущей коллекции (массиве) будут содержаться только те элементы, которые имелись либо в исходном объекте, либо в переданном, но не одновременно в них обоих.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void SymmetricExceptWith(
    ICollection<T>* collection      // коллекция
);
```

Версия для работы с массивом.

```
void SymmetricExceptWith(
    T& array[]                      // массив
);
```

### Параметры

`*collection`

[in] Коллекция, с которой будет строиться симметричная разность.

`&collection[]`

[in] Массив, с которым будет строиться симметричная разность.

### Примечание

Результат записывается в текущую коллекцию (массив).

## UnionWith

Выполняет операцию объединения текущей и переданной коллекции (массива). То есть добавляет в текущую коллекцию (массив) отсутствующие элементы из указанной коллекции (массива).

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void UnionWith(
    ICollection<T>* collection      // коллекция
);
```

Версия для работы с массивом.

```
void UnionWith(
    T& array[]                      // массив
);
```

### Параметры

`*collection`

[in] Коллекция, с которой будет строиться сумма.

`&collection[]`

[in] Массив, с которым будет строиться сумма.

### Примечание

Результат записывается в текущую коллекцию (массив).

## IsProperSubsetOf

Определяет, является ли текущее множество строгим подмножеством заданной коллекции или массива.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool IsProperSubsetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

Версия для работы с массивом.

```
bool IsProperSubsetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true` в случае, если текущее множество является строгим подмножеством, иначе `false`.

## IsProperSupersetOf

Определяет, является ли текущее множество строгим надмножеством заданной коллекции или массива.

Версия для работы с коллекцией реализующей интерфейс `ICollection<T>`.

```
bool IsProperSupersetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

Версия для работы с массивом.

```
bool IsProperSupersetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true`, если текущее множество является строгим надмножеством, иначе `false`.

## IsSubsetOf

Определяет, является ли текущее множество подмножеством заданной коллекции или массива.

**Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.**

```
bool IsSubsetOf(  
    ICollection<T>* collection      // коллекция для определения отношения  
);
```

**Версия для работы с массивом.**

```
bool IsSubsetOf(  
    T& array[]                  // массив для определения отношения  
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true`, если текущее множество является подмножеством, иначе `false`.

## IsSupersetOf

Определяет, является ли текущее множество надмножеством заданной коллекции или массива.

**Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.**

```
bool IsSupersetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

**Версия для работы с массивом.**

```
bool IsSupersetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true` в случае, если текущее множество является надмножеством, иначе `false`.

## Overlaps

Определяет, пересекается ли текущее множество с заданной коллекцией или массивом.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool Overlaps(
    ICollection<T>* collection      // коллекция для сравнения
);
```

Версия для работы с массивом.

```
bool Overlaps(
    T& array[]                  // массив для сравнения
);
```

### Параметры

`*collection`

[in] Коллекция для определения пересечения.

`&collection[]`

[in] Массив для определения пересечения.

### Возвращаемое значение

Возвращает `true` в случае, если между текущим множеством и коллекцией или массивом есть пересечение, иначе `false`.

## SetEquals

Определяет, содержит ли текущее множество все элементы из заданной коллекции или массива.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool SetEquals(
    ICollection<T>* collection      // коллекция для сравнения
);
```

Версия для работы с массивом.

```
bool SetEquals(
    T& array[]                  // массив для сравнения
);
```

### Параметры

`*collection`

[in] Коллекция для сопоставления элементов.

`&collection[]`

[in] Коллекция для сопоставления элементов.

### Возвращаемое значение

Возвращает `true` в случае, если в текущем множестве есть все элементы указанной коллекции или массива, иначе `false`.

## CDefaultComparer<T>

Класс CDefaultComparer<T> — вспомогательный класс, который реализует шаблонный интерфейс IComparer<T> на основе глобальных методов Compare.

### Описание

Класс CDefaultComparer<T> используется по умолчанию в шаблонных коллекциях данных, если пользователь явно не использует другой класс, реализующий интерфейс IComparer<T>.

### Декларация

```
template<typename T>
class CDefaultComparer : public IComparer<T>
```

### Заголовок

```
#include <Generic\Internal\DefaultComparer.mqh>
```

### Иерархия наследования

[IComparer](#)

CDefaultComparer

### Методы класса

Метод	Описание
<a href="#">Compare</a>	Сравнивает два значения типа Т

## Compare

Сравнивает два значения типа Т.

```
int Compare(
    Т   x,      // первое значение
    Т   y,      // второе значение
);
```

### Параметры

*x*

[in] Первое значение для сравнения.

*y*

[in] Второе значение для сравнения.

### Возвращаемое значение

Возвращает число, выражающее отношение двух сравниемых значений:

- результат меньше нуля – *x* меньше *y* (*x*<*y*)
- результат равен нулю – *x* равен *y* (*x*=*y*)
- результат больше нуля – *x* больше *y* (*x*>*y*)

### Примечание

Сравнение значений *x* и *y* происходит на основе одной из перегрузок глобального метода Compare, в зависимости от типа Т.

## CDefaultEqualityComparer<T>

Класс CDefaultEqualityComparer<T> – вспомогательный класс, который реализует шаблонный интерфейс IEqualityComparer<T> на основе глобальных методов Equals<T> и GetHashCode.

### Описание

Класс CDefaultEqualityComparer<T> используется по умолчанию в шаблонных коллекциях данных, если пользователь явно не использует другой класс, реализующий интерфейс IEqualityComparer<T>.

### Декларация

```
template<typename T>
class CDefaultEqualityComparer : public IEqualityComparer<T>
```

### Заголовок

```
#include <Generic\Internal\DefaultEqualityComparer.mqh>
```

### Иерархия наследования

```
IEqualityComparer
CDefaultEqualityComparer
```

### Методы класса

Метод	Описание
<a href="#">Equals</a>	Сравнивает два значения типа Т
<a href="#">HashCode</a>	Вычисляет значение хэш-кода от объекта типа Т

## Equals

Сравнивает два значения типа T.

```
bool Equals(  
    T  x,      // первое значение  
    T  y,      // второе значение  
);
```

### Параметры

x

[in] Первое значение для сравнения.

y

[in] Второе значение для сравнения.

### Возвращаемое значение

Возвращает true в если значения равны, иначе false.

## HashCode

Вычисляет значение хэш-кода от объекта типа Т.

```
int HashCode(  
    T  value      // объект для вычисления  
);
```

### Параметры

*value*

[in] Объект для которого нужно получить хэш-код.

### Возвращаемое значение

Возвращает хэш-код.

## CRedBlackTreeNode<T>

Класс CRedBlackTreeNode<T> — вспомогательный класс, необходимый для реализации класса CRedBlackTree<T>.

### Описание

Класс CRedBlackTreeNode<T> — узел красно-черного дерева CRedBlackTree<T>. В этом классе реализованы методы для навигации по дереву.

### Декларация

```
template<typename T>
class CRedBlackTreeNode
```

### Заголовок

```
#include <Generic\RedBlackTree.mqh>
```

### Методы класса

Метод	Описание
<a href="#"><u>Value</u></a>	Возвращает и устанавливает значение узла
<a href="#"><u>Parent</u></a>	Возвращает и устанавливает указатель на родительский узел
<a href="#"><u>Left</u></a>	Возвращает и устанавливает указатель на левый узел
<a href="#"><u>Right</u></a>	Возвращает и устанавливает указатель на правый узел
<a href="#"><u>Color</u></a>	Возвращает и устанавливает цвет узла
<a href="#"><u>IsLeaf</u></a>	Определяет, является ли данный узел листом
<a href="#"><u>CreateEmptyNode</u></a>	Создаёт новый узел черного цвета без предка и потомков и возвращает указатель на него

## Value (метод Get)

Возвращает значение узла.

```
T Value();
```

### Возвращаемое значение

Возвращает значение узла.

## Value (метод Set)

Устанавливает значение узла.

```
void Value(  
    T value // значение узла  
) ;
```

### Параметры

*value*

[in] Значение узла.

## Parent (метод Get)

Возвращает указатель на родительский узел.

```
CRedBlackTreeNode<T>* Parent();
```

### Возвращаемое значение

Возвращает указатель на родительский узел.

## Parent (метод Set)

Устанавливает указатель на родительский узел.

```
void Parent(  
    CRedBlackTreeNode<T>* node // указатель на родительский узел  
) ;
```

### Параметры

*\*node*

[in] Указатель на родительский узел.

## Left (метод Get)

Возвращает указатель на левый узел.

```
CRedBlackTreeNode<T>* Left();
```

### Возвращаемое значение

Возвращает указатель на левый узел.

## Left (метод Set)

Устанавливает указатель на левый узел.

```
void Left(
    CRedBlackTreeNode<T>* node      // указатель на левый узел
);
```

### Параметры

*\*node*

[in] Указатель на левый узел.

## Right (метод Get)

Возвращает указатель на правый узел.

```
CRedBlackTreeNode<T>* Right();
```

### Возвращаемое значение

Возвращает указатель на правый узел.

## Right (метод Set)

Устанавливает указатель на правый узел.

```
void Right(
    CRedBlackTreeNode<T>* node      // указатель на правый узел
);
```

### Параметры

*\*node*

[in] Указатель на правый узел.

## Color (метод Get)

Возвращает цвет узла.

```
ENUM_RED_BLACK_TREE_NODE_TYPE Color();
```

### Возвращаемое значение

Возвращает цвет узла.

## Color (метод Set)

Устанавливает цвет узла.

```
void Color(
    ENUM_RED_BLACK_TREE_NODE_TYPE  clr      // цвет узла
);
```

### Параметры

*clr*

[in] Цвет узла.

### Примечание

Цвет узла задаётся значением из перечисления ENUM\_RED\_BLACK\_TREE\_NODE\_TYPE и бывает двух типов:

- RED\_BLACK\_TREE\_NODE\_RED – красный цвет узла;
- RED\_BLACK\_TREE\_NODE\_BLACK – черный цвет узла.

## IsLeaf

Определяет, является ли данный узел листом.

```
bool IsLeaf();
```

### Возвращаемое значение

Возвращает true, если узел является листом, иначе false.

## CreateEmptyNode

Создаёт новый узел черного цвета без предка и потомков и возвращает указатель на него.

```
static CRedBlackTreeNode<T>* CreateEmptyNode();
```

### Возвращаемое значение

Возвращает указатель на новый узел.

## CLinkedListNode<T>

Класс CLinkedListNode<T> — вспомогательный класс, необходимый для реализации класса CLinkedListNode<T>.

### Описание

Класс CLinkedListNode<T> является узлом двунаправленного списка CLinkedListNode<T>. В классе реализованы методы для навигации по списку.

### Декларация

```
template<typename T>
class CLinkedListNode
```

### Заголовок

```
#include <Generic\LinkedList.mqh>
```

### Методы класса

Метод	Описание
<u>List</u>	Возвращает и устанавливает указатель на двунаправленный список CLinkedList<T>
<u>Next</u>	Возвращает и устанавливает указатель на следующий узел
<u>Previous</u>	Возвращает и устанавливает указатель на предыдущий узел
<u>Value</u>	Возвращает и устанавливает значение узла

## List (метод Get)

Возвращает указатель на двунаправленный список CLinkedList<T>.

```
CLinkedList<T>* List();
```

### Возвращаемое значение

Возвращает указатель на двунаправленный список CLinkedList<T>.

## List (метод Set)

Устанавливает указатель на двунаправленный список CLinkedList<T>.

```
void List(
    CLinkedList<T>* value      // указатель на список
);
```

### Параметры

\*value

[in] Указатель на двунаправленный список CLinkedList<T>.

## Next (метод Get)

Возвращает указатель на следующий узел.

```
CLinkedListNode<T>* Next();
```

### Возвращаемое значение

Возвращает указатель на следующий узел.

## Next (метод Set)

Устанавливает указатель на следующий узел.

```
void Next(
    CLinkedListNode<T>* value      // указатель на следующий узел
);
```

### Параметры

*\*value*

[in] Указатель на следующий узел.

## Previous (метод Get)

Возвращает указатель на предыдущий узел.

```
CLinkedListNode<T>* Previous();
```

### Возвращаемое значение

Возвращает указатель на предыдущий узел.

## Previous (метод Set)

Устанавливает указатель на предыдущий узел.

```
void Previous(
    CLinkedListNode<T>* value      // указатель на предыдущий узел
);
```

### Параметры

*\*value*

[in] Указатель на предыдущий узел.

## Value (метод Get)

Возвращает значение узла.

```
T Value();
```

### Возвращаемое значение

Возвращает значение узла.

## Value (метод Set)

Устанавливает значение узла.

```
void Value(  
    T value // значение узла  
) ;
```

### Параметры

*value*

[in] Значение узла.

## CKeyValuePair< TKey, TValue >

Класс CKeyValuePair< TKey, TValue > – класс реализует пару "ключ – значение".

### Описание

Класс CKeyValuePair< TKey, TValue > реализует методы непосредственной работы с ключом и значением пары "ключ – значение".

### Декларация

```
template<typename TKey, typename TValue>
class CKeyValuePair : public IComparable<CKeyValuePair<TKey, TValue>*>
```

### Заголовок

```
#include <Generic\HashMap.mqh>
```

### Иерархия наследования

[IEqualityComparable](#)

[IComparable](#)

CKeyValuePair

### Методы класса

Метод	Описание
<a href="#">Key</a>	Возвращает и устанавливает ключ пары "ключ – значение"
<a href="#">Value</a>	Возвращает и устанавливает значение пары "ключ – значение"
<a href="#">Clone</a>	Создаёт новую пару "ключ – значение", ключ и значение которой равны текущим
<a href="#">Compare</a>	Сравнивает текущую пару "ключ – значение" с указанной
<a href="#">Equals</a>	Сравнивает текущую пару "ключ – значение" с указанной на предмет равенства
<a href="#">HashCode</a>	Вычисляет значение хэш-кода от пары "ключ – значение"

## Key (метод Get)

Возвращает ключ пары "ключ — значение".

```
TKey Key();
```

### Возвращаемое значение

Возвращает ключ.

## Key (метод Set)

Устанавливает значение ключа пары "ключ — значение".

```
void Key(
    TKey key      // ключ
);
```

### Параметры

*key*

[in] Ключ.

## Value (метод Get)

Возвращает значение пары "ключ — значение".

```
TValue Value();
```

### Возвращаемое значение

Возвращает значение.

## Value (метод Set)

Устанавливает значение пары "ключ — значение".

```
void Value(
    TValue  value      // значение
);
```

### Параметры

*value*

[in] Значение

## Clone

Создаёт новую пару "ключ — значение", ключ и значение которой равны текущим.

```
TValue>* Clone();
```

### Возвращаемое значение

Возвращает новую пару "ключ — значение"

## Compare

Сравнивает текущую пару "ключ — значение" с указанной.

```
int Compare(
    CKeyValuePair<TKey TValue>*< i>pair      // пара для сравнения
);
```

### Параметры

*\*pair*

[in] Пары для сравнения.

### Возвращаемое значение

Возвращает число, выражающее отношение текущей и переданной пары "ключ — значение":

- результат меньше нуля — текущая пара "ключ-значение" меньше переданной
- результат равен нулю — текущая пара "ключ-значение" равна переданной
- результат больше нуля — текущая пара "ключ-значение" больше переданной

### Примечание

Сравнение двух пар "ключ — значение" происходит по ключу.

## Equals

Сравнивает текущую пару "ключ — значение" с указанной на предмет равенства.

```
bool Equals(  
    CKeyValuePair<TKey TValue>*<br/>        pair      // пара для сравнения  
);
```

### Параметры

*\*pair*

[in] Пары для сравнения

### Возвращаемое значение

Возвращает true, если пары "ключ — значение" равны, иначе false.

### Примечание

Сравнение двух пар "ключ — значение" происходит по ключу.

## HashCode

Вычисляет значение хэш-кода от пары "ключ — значение".

```
int HashCode();
```

### Возвращаемое значение

Возвращает хэш-код.

### Примечание

Хэш-код пары "ключ — значение" равен хэш-коду ключа.

## CArrayList<T>

Класс CArrayList<T> – шаблонный класс, реализующий интерфейс IList<T>.

### Описание

Класс CArrayList<T> является реализацией динамического списка данных типа T. Данный класс предоставляет основные методы для работы со списком: доступ к элементу по индексу, поиск и удаление элемента, сортировку и другие.

### Декларация

```
template<typename T>
class CArrayList : public IList<T>
```

### Заголовок

```
#include <Generic\ArrayList.mqh>
```

### Иерархия наследования

```
ICollection
  IList
    CArrayList
```

### Методы класса

Метод	Описание
<a href="#">Capacity</a>	Возвращает и устанавливает текущую емкость списка
<a href="#">Count</a>	Возвращает количество элементов в списке
<a href="#">Contains</a>	Определяет, содержит ли список элемент с указанным значением
<a href="#">TrimExcess</a>	Сокращает емкость списка до фактического числа элементов
<a href="#">TryGetValue</a>	Получает элемент списка по указанному индексу
<a href="#">TrySetValue</a>	Устанавливает значение элемента списка по указанному индексу
<a href="#">Add</a>	Добавляет элемент в список
<a href="#">AddRange</a>	Добавляет коллекцию или массив элементов в список
<a href="#">Insert</a>	Вставляет элемент в список по указанному индексу

<a href="#"><u>InsertRange</u></a>	Вставляет коллекцию или массив элементов в список по указанному индексу
<a href="#"><u>CopyTo</u></a>	Копирует все элементы списка в указанный массив, начиная с определенного индекса
<a href="#"><u>BinarySearch</u></a>	Ищет указанное значение в отсортированном по возрастанию списке
<a href="#"><u>IndexOf</u></a>	Ищет первое вхождение значения в списке
<a href="#"><u>LastIndexOf</u></a>	Ищет последнее вхождение значения в списке
<a href="#"><u>Clear</u></a>	Удаляет все элементы коллекции
<a href="#"><u>Remove</u></a>	Удаляет первое вхождение указанного элемента из списка
<a href="#"><u>RemoveAt</u></a>	Удаляет указанный элемент из списка по заданному индексу
<a href="#"><u>RemoveRange</u></a>	Удаляет диапазон элементов из списка
<a href="#"><u>Reverse</u></a>	Изменяет последовательность элементов в списке
<a href="#"><u>Sort</u></a>	Выполняет сортировку элементов списка

## Capacity (метод Get)

Возвращает текущую емкость списка.

```
int Capacity();
```

### Возвращаемое значение

Возвращает текущую емкость списка.

## Capacity (метод Set)

Устанавливает текущую емкость списка.

```
void Capacity(  
    const int capacity // значение емкости  
) ;
```

### Параметры

*capacity*

[in] Новое значение емкости.

## Count

Возвращает количество элементов в списке.

```
int Count();
```

### Возвращаемое значение

Возвращает количество элементов.

## Contains

Определяет, содержит ли список элемент с указанным значением.

```
bool Contains(  
    T   item      // искомое значение  
);
```

### Параметры

*item*

[in] Искомое значение.

### Возвращаемое значение

Возвращает true, если в списке есть элемент с указанным значением, иначе false.

## TrimExcess

Сокращает емкость списка до фактического числа элементов, тем самым освобождая неиспользованную память.

```
void TrimExcess();
```

## TryGetValue

Получает элемент списка по указанному индексу.

```
bool TryGetValue(
    const int    index,      // индекс
    T&          value       // переменная для записи
);
```

### Параметры

*index*

[in] Индекс элемента списка, значение которого необходимо получить.

*&value*

[out] Переменная для записи значения элемента.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## TrySetValue

Устанавливает значение элемента списка по указанному индексу.

```
bool TrySetValue(
    const int   index,      // индекс
    T          value       // значение элемента
);
```

### Параметры

*index*

[in] Индекс элемента списка, значение которого необходимо установить.

*value*

[in] Устанавливаемое значение элемента списка.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Add

Добавляет элемент в список.

```
bool Add(  
    T   value      // значение элемента  
);
```

### Параметры

*value*

[in] Значение элемента для добавления.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## AddRange

Добавляет коллекцию или массив элементов в список.

**Версия для добавления массива.**

```
bool AddRange(
    const T& array[]           // массив для добавления
);
```

**Версия для добавления коллекции.**

```
bool AddRange(
    ICollection<T>* collection // коллекция для добавления
);
```

**Параметры**

*&array[]*

[in] Массив для добавления.

*\*collection*

[in] Коллекция для добавления.

**Возвращаемое значение**

Возвращает true в случае успеха, иначе false.

## Insert

Вставляет элемент в список по указанному индексу.

```
bool Insert(  
    const int   index,      // индекс для вставки  
    T          item        // значение для вставки  
);
```

### Параметры

*index*

[in] Индекс для вставки.

*item*

[in] Значение, которое необходимо вставить по указанному индексу.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## InsertRange

Вставляет коллекцию или массив элементов в список по указанному индексу.

**Версия для вставки массива.**

```
bool InsertRange(
    const int    index,           // индекс для вставки
    const T&     array[]         // массив для вставки
);
```

**Версия для вставки коллекции.**

```
bool InsertRange(
    const int      index,        // индекс для вставки
    ICollection<T>* collection  // коллекция для вставки
);
```

### Параметры

*index*

[in] Индекс для вставки.

*&array[]*

[in] Массив, который необходимо вставить по указанному индексу.

*\*collection*

[in] Коллекция, которую необходимо вставить по указанному индексу.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## CopyTo

Копирует все элементы списка в указанный массив, начиная с определенного индекса.

```
int CopyTo(  
    T&          dst_array[],      // массив для записи  
    const int   dst_start=0        // начальный индекс для записи  
);
```

### Параметры

`&dst_array[]`

[out] Массив, в который будут записаны элементы списка.

`dst_start=0`

[in] Индекс в массиве, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных элементов.

## BinarySearch

Ищет указанное значение в отсортированном по возрастанию списке.

Версия для поиска по заданному диапазону значений и с использованием класса, реализующего интерфейс `IComparable<T>` для сравнения элементов.

```
int BinarySearch(
    const int      index,          // начальный индекс
    const int      count,          // диапазон поиска
    T              item,           // искомое значение
    IComparer<T>* comparer       // интерфейс для сравнения
);
```

Версия для поиска с использованием класса, реализующего интерфейс `IComparable<T>` для сравнения элементов.

```
int BinarySearch(
    T              item,           // искомое значение
    IComparer<T>* comparer       // интерфейс для сравнения
);
```

Версия для поиска с использованием глобального метода `::Compare` для сравнения элементов.

```
int BinarySearch(
    T  item                   // искомое значение
);
```

### Параметры

*index*

[in] Начальный индекс, с которого начинается поиск.

*count*

[in] Длина диапазона поиска.

*item*

[in] Искомое значение.

*\*comparer*

[in] Интерфейс для сравнения элементов.

### Возвращаемое значение

Возвращает индекс найденного элемента. Если искомое значение не найдено, то возвращает индекс меньшего элемента, которое ближе всех по значению.

## IndexOf

Ищет первое вхождение значения в списке.

**Версия для поиска по всему списку.**

```
int IndexOf(
    T item           // искомое значение
);
```

**Версия для поиска с указанной позиции и до конца списка.**

```
int IndexOf(
    T item,           // искомое значение
    const int start_index // начальный индекс
);
```

**Версия для поиска с указанной позиции в заданном диапазоне.**

```
int IndexOf(
    T item,           // искомое значение
    const int start_index, // начальный индекс
    const int count      // диапазон поиска
);
```

### Параметры

*item*

[in] Искомое значение.

*start\_index*

[in] Начальный индекс, с которого начинается поиск.

*count*

[in] Длина диапазона поиска.

### Возвращаемое значение

Возвращает индекс первого найденного элемента. Если значение не найдено, возвращает -1.

## LastIndexOf

Ищет последнее вхождение значения в списке.

**Версия для поиска по всему списку.**

```
int LastIndexOf(
    T item           // искомое значение
);
```

**Версия для поиска с указанной позиции и до конца списка.**

```
int LastIndexOf(
    T item,           // искомое значение
    const int start_index // начальный индекс
);
```

**Версия для поиска с указанной позиции в заданном диапазоне.**

```
int LastIndexOf(
    T item,           // искомое значение
    const int start_index, // начальный индекс
    const int count      // диапазон поиска
);
```

### Параметры

*item*

[in] Искомое значение.

*start\_index*

[in] Начальный индекс, с которого начинается поиск.

*count*

[in] Длина диапазона поиска.

### Возвращаемое значение

Возвращает индекс последнего найденного элемента. Если значение не найдено, возвращает -1.

## Clear

Удаляет все элементы коллекции.

```
void Clear();
```

## Remove

Удаляет первое вхождение указанного элемента из списка.

```
bool Remove(  
    T    item      // значение элемента  
);
```

### Параметры

*item*

[in] Значение элемента, которое нужно удалить.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## RemoveAt

Удаляет указанный элемент из списка по заданному индексу.

```
bool RemoveAt(
    const int   index      // индекс
);
```

### Параметры

*index*

[in] Индекс удаляемого элемента.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## RemoveRange

Удаляет диапазон элементов из списка.

```
bool RemoveRange(
    const int    start_index,      // начальный индекс
    const int    count            // количество элементов
);
```

### Параметры

*start\_index*

[in] Начальный индекс, с которого начинается удаление.

*count*

[in] Количество элементов, подлежащих удалению.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Reverse

Изменяет последовательность элементов в списке.

Версия для работы со всем списком.

```
bool Reverse();
```

Версия для работы с заданным диапазоном элементов списка.

```
bool Reverse(  
    const int    start_index,      // начальный индекс  
    const int    count            // количество элементов  
);
```

### Параметры

*start\_index*

[in] Начальный индекс.

*count*

[in] Количество элементов списка, участвующих в операции.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Sort

Выполняет сортировку элементов списка.

**Версия для сортировки всех элементов списка.**

```
bool Sort();
```

**Версия для сортировки всех элементов списка с использованием класса, реализующего интерфейс `IComparable<T>` для сравнения элементов.**

```
bool Sort(
    IComparer<T>* comparer           // интерфейс для сравнения
);
```

**Версия для сортировки заданного диапазона элементов списка с использованием класса, реализующего интерфейс `IComparable<T>` для сравнения элементов.**

```
bool Sort(
    const int      start_index,        // начальный индекс
    const int      count,              // количество элементов
    IComparer<T>* comparer           // интерфейс для сравнения
);
```

### Параметры

`*comparer`

[in] Интерфейс для сравнения элементов.

`start_index`

[in] Начальный индекс, с которого начинается сортировка.

`count`

[in] Длина диапазона сортировки.

### Возвращаемое значение

Возвращает `true` в случае успеха, иначе `false`.

## CHashMap<TKey, TValue>

Класс CHashMap<TKey, TValue> – шаблонный класс, реализующий интерфейс IMap<TKey, TValue>.

### Описание

Класс CHashMap<TKey, TValue> является реализацией динамической хэш-таблицы, данные которой хранятся в виде неупорядоченных пар "ключ – значение" с соблюдением требования уникальности ключа. Данный класс предоставляет основные методы для работы с хэш-таблицей: доступ к значению по ключу, поиск и удаление пары "ключ-значение" и другие.

### Декларация

```
template<typename TKey, typename TValue>
class CHashMap : public IMap<TKey, TValue>
```

### Заголовок

```
#include <Generic\HashMap.mqh>
```

### Иерархия наследования

[ICollection](#)

[IMap](#)

CHashMap

### Методы класса

Метод	Описание
<a href="#">Add</a>	Добавляет пару "ключ – значение" в хэш-таблицу
<a href="#">Count</a>	Возвращает количество элементов в хэш-таблице
<a href="#">Comparer</a>	Возвращает указатель на интерфейс IEqualityComparer<T>, использующийся для организации хэш-таблицы
<a href="#">Contains</a>	Определяет, содержит ли хэш-таблица указанную пару "ключ – значение"
<a href="#">ContainsKey</a>	Определяет, содержит ли хэш-таблица пару "ключ – значение" с указанным ключом
<a href="#">ContainsValue</a>	Класс CHashMap<TKey, TValue> – шаблонный класс, реализующий интерфейс IMap<TKey, TValue>
<a href="#">CopyTo</a>	Копирует все пары "ключ – значение" из хэш-таблицы в указанные массивы, начиная с определенного индекса

<a href="#"><u>Clear</u></a>	Удаляет все элементы хэш-таблицы
<a href="#"><u>Remove</u></a>	Удаляет первое вхождение пары "ключ — значение" хэш-таблицы
<a href="#"><u>TryGetValue</u></a>	Получает элемент из хэш-таблицы по заданному ключу
<a href="#"><u>TrySetValue</u></a>	Изменяет значение пары "ключ — значение" из хэш-таблицы по заданному ключу

## Add

Добавляет пару "ключ – значение" в хэш-таблицу.

Версия для добавления сформированной пары "ключ – значение".

```
bool Add(  
    CKeyValuePair< TKey TValue>* pair // пара "ключ – значение"  
) ;
```

Версия для добавления новой пары "ключ – значение" с указанным ключом и значением.

```
bool Add(  
    TKey key, // ключ  
    TValue value // значение  
) ;
```

### Параметры

*\*pair*

[in] Пары "ключ – значение".

*key*

[in] Ключ.

*value*

[in] Значение.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Count

Возвращает количество элементов в хэш-таблице.

```
int Count();
```

## Comparer

Возвращает указатель на интерфейс `IEqualityComparer<T>`, использующийся для организации хэш-таблицы.

```
IEqualityComparer<TKey>* Comparer() const;
```

### Возвращаемое значение

Возвращает указатель на интерфейс `IEqualityComparer<T>`.

## Contains

Определяет, содержит ли хэш-таблица указанную пару "ключ — значение".

Версия для работы с сформированной парой "ключ — значение".

```
bool Contains(  
    CKeyValuePair< TKey TValue>* item // пара "ключ — значение"  
) ;
```

Версия для работы с парой "ключ — значение" в виде отдельно заданного ключа и значения.

```
bool Contains(  
    TKey key, // ключ  
    TValue value // значение  
) ;
```

### Параметры

*\*item*  
[in] Пара "ключ — значение".

*key*  
[in] Ключ.

*value*  
[in] Значение.

### Возвращаемое значение

Возвращает true, если в хэш-таблице есть пара "ключ-значение" с указанным ключом и значением, иначе false.

## ContainsKey

Определяет, содержит ли хэш-таблица пару "ключ — значение" с указанным ключом.

```
bool ContainsKey(  
    TKey key // ключ  
) ;
```

### Параметры

*key*

[in] Ключ.

### Возвращаемое значение

Возвращает true если в хэш-таблице есть пара "ключ — значение" с указанным ключом, иначе false.

## ContainsValue

Определяет, содержит ли хэш-таблица пару "ключ — значение" с указанным значением.

```
bool ContainsValue(  
    TValue  value      // значение  
);
```

### Параметры

*value*

[in] Значение.

### Возвращаемое значение

Возвращает true, если в хэш-таблице есть пара "ключ — значение" с указанным значением, иначе false.

## CopyTo

Копирует все пары "ключ – значение" из хэш-таблицы в указанные массивы, начиная с определенного индекса.

**Версия для копирования хэш-таблицы в массив пар "ключ – значение".**

```
int CopyTo(
    CKeyValuePair< TKey TValue>*& dst_array[],           // массив для записи пар "ключ – значе
    const int                      dst_start=0          // начальный индекс для записи
);
```

**Версия для копирования хэш-таблицы в отдельные массивы для ключей и значений.**

```
int CopyTo(
    TKey&      dst_keys[],           // массив для записи ключей
    TValue&    dst_values[],        // массив для записи значений
    const int   dst_start=0          // начальный индекс для записи
);
```

### Параметры

`*&dst_array[]`

[out] Массив, в который будут записаны все пары из хэш-таблицы.

`&dst_keys[]`

[out] Массив, в который будут записаны все ключи из хэш-таблицы.

`&dst_values[]`

[out] Массив, в который будут записаны все значения из хэш-таблицы.

`dst_start=0`

[in] Индекс массива, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных пар "ключ – значение".

## Clear

Удаляет все элементы хэш-таблицы.

```
void Clear();
```

## Remove

Удаляет первое вхождение пары "ключ — значение" хэш-таблицы.

Версия для удаления пары "ключ — значение" по сформированной паре "ключ — значение".

```
bool Remove(  
    CKeyValuePair< TKey TValue>*& item // пара "ключ-значение"  
) ;
```

Версия для удаления пары "ключ — значение" по ключу.

```
bool Remove(  
    TKey key // ключ  
) ;
```

### Параметры

*\*item*  
[in] Пары "ключ — значение".

*key*  
[in] Ключ.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## TryGetValue

Получает элемент из хэш-таблицы по заданному ключу.

```
bool TryGetValue(  
    TKey      key,          // ключ  
    TValue&   value        // переменная для записи значения  
);
```

### Параметры

*key*

[in] Ключ.

*&value*

[out] Переменная, в которую будет записано указанное значение пары "ключ — значение".

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## TrySetValue

Изменяет значение пары "ключ — значение" из хэш-таблицы по заданному ключу.

```
bool TrySetValue(  
    TKey     key,      // Ключ  
    TValue   value     // новое значение  
);
```

### Параметры

*key*

[in] Ключ.

*value*

[in] Новое значение, которое нужно присвоить указанной паре "ключ — значение".

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## CHashSet<T>

Класс CHashSet<T> – шаблонный класс, реализующий интерфейс ISet<T>.

### Описание

Класс CHashSet<T> является реализацией неупорядоченного динамического множества данных типа T, с соблюдением требования уникальности каждого значения. Данный класс предоставляет основные методы для работы с множествами и операциями над ними: объединением и пересечением множеств, определением строгих и нестрогих подмножеств и другие.

### Декларация

```
template<typename T>
class CHashSet : public ISet<T>
```

### Заголовок

```
#include <Generic\HashSet.mqh>
```

### Иерархия наследования

[ICollection](#)

[ISet](#)

CHashSet

### Методы класса

Метод	Описание
<a href="#">Add</a>	Добавляет элемент в множество
<a href="#">Count</a>	Возвращает количество элементов в множестве
<a href="#">Comparer</a>	Определяет, содержит ли множество элемент с указанным значением
<a href="#">Contains</a>	Возвращает указатель на интерфейс IEqualityComparer<T>, использующийся для организации множества
<a href="#">TrimExcess</a>	Сокращает емкость множества до фактического числа элементов, тем самым освобождая неиспользованную память
<a href="#">CopyTo</a>	Копирует все элементы множества в указанный массив, начиная с определенного индекса
<a href="#">Clear</a>	Удаляет все элементы множества
<a href="#">Remove</a>	Удаляет указанный элемент из множества

<a href="#"><u>ExceptWith</u></a>	Выполняет операцию разности текущей и переданной коллекции (массива)
<a href="#"><u>IntersectWith</u></a>	Выполняет операцию пересечения текущей и переданной коллекции (массива)
<a href="#"><u>SymmetricExceptWith</u></a>	Выполняет операцию симметричной разности текущей и переданной коллекции (массива)
<a href="#"><u>UnionWith</u></a>	Выполняет операцию объединения текущей и переданной коллекции (массива)
<a href="#"><u>IsProperSubsetOf</u></a>	Определяет, является ли текущее множество строгим подмножеством заданной коллекции или массива
<a href="#"><u>IsProperSupersetOf</u></a>	Определяет, является ли текущее множество строгим надмножеством заданной коллекции или массива
<a href="#"><u>IsSubsetOf</u></a>	Определяет, является ли текущее множество подмножеством заданной коллекции или массива
<a href="#"><u>IsSupersetOf</u></a>	Определяет, является ли текущее множество надмножеством заданной коллекции или массива
<a href="#"><u>Overlaps</u></a>	Определяет, пересекается ли текущее множество с заданной коллекцией или массивом
<a href="#"><u>SetEquals</u></a>	Определяет, содержит ли текущее множество все элементы из заданной коллекции или массива

## Add

Добавляет элемент в множество.

```
bool Add(  
    T   value      // значение элемента  
);
```

### Параметры

*value*

[in] Значение элемента для добавления.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Count

Возвращает количество элементов в множестве.

```
int Count();
```

### Возвращаемое значение

Возвращает количество элементов.

## Contains

Определяет, содержит ли множество элемент с указанным значением.

```
bool Contains(  
    T   item      // искомое значение  
) ;
```

### Параметры

*item*

[in] Искомое значение.

### Возвращаемое значение

Возвращает true, если во множестве есть элемент с указанным значением, иначе false.

## Comparer

Возвращает указатель на интерфейс `IEqualityComparer<T>`, используемый для организации множества.

```
IEqualityComparer<T>* Comparer() const;
```

### Возвращаемое значение

Возвращает указатель на интерфейс `IEqualityComparer<T>`.

## TrimExcess

Сокращает емкость множества до фактического числа элементов, тем самым освобождая неиспользованную память.

```
void TrimExcess();
```

## CopyTo

Копирует все элементы множества в указанный массив, начиная с определенного индекса.

```
int CopyTo(  
    T&          dst_array[],      // массив для записи  
    const int   dst_start=0        // начальный индекс для записи  
);
```

### Параметры

`&dst_array[]`

[out] Массив, в который будут записаны элементы множества.

`dst_start=0`

[in] Индекс в массиве, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных элементов.

## Clear

Удаляет все элементы множества.

```
void Clear();
```

## Remove

Удаляет указанный элемент из множества.

```
bool Remove(  
    T    item      // значение элемента  
);
```

### Параметры

*item*

[in] Значение элемента, которое нужно удалить.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## ExceptWith

Выполняет операцию разности текущей и переданной коллекции (массива). То есть удаляет из текущей коллекции (массива) все элементы, которые есть также и в указанной коллекции (массиве).

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void ExceptWith(
    ICollection<T>* collection      // коллекция
);
```

Версия для работы с массивом.

```
void ExceptWith(
    T& array[]                  // массив
);
```

### Параметры

`*collection`

[in] Коллекция, которая будет вычитаться из текущего множества.

`&collection[]`

[in] Массив, который будет вычитаться из текущего множества.

### Примечание

Результат записывается в текущую коллекцию (массив).

## IntersectWith

Выполняет операцию пересечения текущей и переданной коллекции (массива). То есть оставляет в текущей коллекции только те элементы, которые имеются также и в указанной коллекции (массиве).

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void IntersectWith(
    ICollection<T>* collection      // коллекция
);
```

Версия для работы с массивом.

```
void IntersectWith(
    T& array[]                      // массив
);
```

### Параметры

`*collection`

[in] Коллекция, с которой будет строиться произведение.

`&collection[]`

[in] Массив, с которым будет строиться произведение.

### Примечание

Результат записывается в текущую коллекцию (массив).

## SymmetricExceptWith

Выполняет операцию симметричной разности текущей и переданной коллекции (массива). То есть в текущей коллекции (массиве) будут содержаться только те элементы, которые имелись либо в исходном объекте, либо в переданном, но не одновременно в них обоих.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void SymmetricExceptWith(
    ICollection<T>* collection      // коллекция
);
```

Версия для работы с массивом.

```
void SymmetricExceptWith(
    T& array[]                      // массив
);
```

### Параметры

`*collection`

[in] Коллекция, с которой будет строиться симметричная разность.

`&collection[]`

[in] Массив, с которым будет строиться симметричная разность.

### Примечание

Результат записывается в текущую коллекцию (массив).

## UnionWith

Выполняет операцию объединения текущей и переданной коллекции (массива). То есть добавляет в текущую коллекцию (массив) отсутствующие элементы из указанной коллекции (массива).

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void UnionWith(  
    ICollection<T>* collection      // коллекция  
) ;
```

Версия для работы с массивом.

```
void UnionWith(  
    T& array[]                      // массив  
) ;
```

### Параметры

`*collection`

[in] Коллекция, с которой будет строиться сумма.

`&collection[]`

[in] Массив, с которым будет строиться сумма.

### Примечание

Результат записывается в текущую коллекцию (массив).

## IsProperSubsetOf

Определяет, является ли текущее множество строгим подмножеством заданной коллекции или массива.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool IsProperSubsetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

Версия для работы с массивом.

```
bool IsProperSubsetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true` в случае, если текущее множество является строгим подмножеством, иначе `false`.

## IsProperSupersetOf

Определяет, является ли текущее множество строгим надмножеством заданной коллекции или массива.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool IsProperSupersetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

Версия для работы с массивом.

```
bool IsProperSupersetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true` в случае, если текущее множество является строгим надмножеством, иначе `false`.

## IsSubsetOf

Определяет, является ли текущее множество подмножеством заданной коллекции или массива.

**Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.**

```
bool IsSubsetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

**Версия для работы с массивом.**

```
bool IsSubsetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true` в случае, если текущее множество является подмножеством, иначе `false`.

## IsSupersetOf

Определяет, является ли текущее множество надмножеством заданной коллекции или массива.

**Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.**

```
bool IsSupersetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

**Версия для работы с массивом.**

```
bool IsSupersetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true` в случае, если текущее множество является надмножеством, иначе `false`.

## Overlaps

Определяет, пересекается ли текущее множество с заданной коллекцией или массивом.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool Overlaps(
    ICollection<T>* collection      // коллекция для сравнения
);
```

Версия для работы с массивом.

```
bool Overlaps(
    T& array[]                  // массив для сравнения
);
```

### Параметры

`*collection`

[in] Коллекция для определения пересечения.

`&collection[]`

[in] Массив для определения пересечения.

### Возвращаемое значение

Возвращает `true` в случае, если между текущим множеством и коллекцией или массивом есть пересечение, иначе `false`.

## SetEquals

Определяет, содержит ли текущее множество все элементы из заданной коллекции или массива.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool SetEquals(
    ICollection<T>* collection      // коллекция для сравнения
);
```

Версия для работы с массивом.

```
bool SetEquals(
    T& array[]                  // массив для сравнения
);
```

### Параметры

`*collection`

[in] Коллекция для сопоставления элементов.

`&collection[]`

[in] Массив для сопоставления элементов.

### Возвращаемое значение

Возвращает `true` в случае, если в текущем множестве есть все элементы указанной коллекции или массива, иначе `false`.

## CLinkedList<T>

Класс CLinkedList<T> – шаблонный класс, реализующий интерфейс ICollection<T>.

### Описание

Класс CLinkedList<T> является реализацией динамического двунаправленного списка данных типа T. Данный класс предоставляет основные методы для работы с двунаправленными списками: добавление, удаление, поиск элемента и другие.

### Декларация

```
template<typename T>
class CLinkedList : public ICollection<T>
```

### Заголовок

```
#include <Generic\LinkedList.mqh>
```

### Иерархия наследования

```
ICollection
CLinkedList
```

### Методы класса

Метод	Описание
<a href="#">Add</a>	Добавляет элемент в двунаправленный список
<a href="#">AddAfter</a>	Добавляет элемент в двунаправленный список после указанного узла
<a href="#">AddBefore</a>	Добавляет элемент в двунаправленный список до указанного узла
<a href="#">AddFirst</a>	Добавляет элемент в начало двунаправленного списка
<a href="#">AddLast</a>	Добавляет элемент в конец двунаправленного списка
<a href="#">Count</a>	Возвращает количество элементов в двунаправленном списке
<a href="#">Head</a>	Возвращает указатель на первый узел двунаправленного списка
<a href="#">First</a>	Возвращает указатель на первый узел двунаправленного списка
<a href="#">Last</a>	Возвращает указатель на последний узел двунаправленного списка

<a href="#"><u>Contains</u></a>	Определяет, содержит ли двунаправленный список элемент с указанным значением
<a href="#"><u>CopyTo</u></a>	Копирует все элементы двунаправленного списка в указанный массив, начиная с определенного индекса
<a href="#"><u>Clear</u></a>	Удаляет все элементы коллекции
<a href="#"><u>Remove</u></a>	Удаляет первое вхождение указанного элемента из двунаправленного списка
<a href="#"><u>RemoveFirst</u></a>	Удаляет первый элемент из двунаправленного списка
<a href="#"><u>RemoveLast</u></a>	Удаляет последний элемент из двунаправленного списка
<a href="#"><u>Find</u></a>	Ищет первое вхождение заданного значения в двунаправленный список
<a href="#"><u>FindLast</u></a>	Ищет последнее вхождение заданного значения в двунаправленный список

## Add

Добавляет элемент в двунаправленный список.

```
bool Add(  
    T   value      // значение элемента  
);
```

### Параметры

*value*

[in] Значение элемента для добавления.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## AddAfter

Добавляет элемент в двунаправленный список после указанного узла.

**Версия для добавления элемента по значению.**

```
CLinkedListNode<T>* AddAfter(
    CLinkedListNode<T>* node,           // узел, после которого следует добавление
    T                      value        // добавляемый элемент
);
```

**Возвращаемое значение**

Возвращает указатель на добавленный узел.

**Версия для добавления элемента как сформированного узла по значению.**

```
bool AddAfter(
    CLinkedListNode<T>* node,           // узел, после которого следует добавление
    CLinkedListNode<T>* new_node        // добавляемый узел
);
```

**Параметры**

*\*node*

[in] Узел двунаправленного списка, после которого будет добавлен новый элемент.

*value*

[in] Элемент для добавления.

*\*new\_node*

[in] Узел для добавления.

**Возвращаемое значение**

Возвращает true в случае успеха, иначе false.

## AddBefore

Добавляет элемент в двунаправленный список до указанного узла.

**Версия для добавления элемента по значению.**

```
CLinkedListNode<T>* AddBefore(
    CLinkedListNode<T>* node,           // узел, до которого следует добавление
    T                      value        // добавляемый элемент
);
```

**Возвращаемое значение**

Возвращает указатель на добавленный узел.

**Версия для добавления элемента как сформированного узла по значению.**

```
bool AddBefore(
    CLinkedListNode<T>* node,           // узел, до которого следует добавление
    CLinkedListNode<T>* new_node        // добавляемый узел
);
```

**Параметры**

*\*node*

[in] Узел двунаправленного списка, до которого будет добавлен новый элемент.

*value*

[in] Элемент для добавления.

*\*new\_node*

[in] Узел для добавления.

**Возвращаемое значение**

Возвращает true в случае успеха, иначе false.

## AddFirst

Добавляет элемент в начало двунаправленного списка.

**Версия для добавления элемента по значению.**

```
CLinkedListNode<T>* AddFirst(
    T     value           // элемент для добавления
);
```

**Возвращаемое значение**

Возвращает указатель на добавленный узел.

**Версия для добавления элемента как сформированного узла по значению.**

```
bool AddFirst(
    CLinkedListNode<T>* node      // узел для добавления
);
```

**Параметры**

*value*

[in] Элемент для добавления.

*\*node*

[in] Узел для добавления.

**Возвращаемое значение**

Возвращает true в случае успеха, иначе false.

## AddLast

Добавляет элемент в конец двунаправленного списка.

**Версия для добавления элемента по значению.**

```
CLinkedListNode<T>* AddLast (
    T   value           // элемент для добавления
);
```

**Возвращаемое значение**

Возвращает указатель на добавленный узел.

**Версия для добавления элемента, как сформированного узла по значению.**

```
bool AddLast (
    CLinkedListNode<T>*  node      // узел для добавления
);
```

**Параметры**

*value*

[in] Элемент для добавления.

*\*node*

[in] Узел для добавления.

**Возвращаемое значение**

Возвращает true в случае успеха, иначе false.

## Count

Возвращает количество элементов в двунаправленном списке.

```
int Count();
```

### Возвращаемое значение

Возвращает количество элементов.

## Head

Возвращает указатель на первый узел двунаправленного списка.

```
CLinkedListNode<T>* Head();
```

### Возвращаемое значение

Возвращает указатель на первый узел.

## First

Возвращает указатель на первый узел двунаправленного списка.

```
CLinkedListNode<T>* First();
```

### Возвращаемое значение

Возвращает указатель на первый узел.

## Last

Возвращает указатель на последний узел двунаправленного списка.

```
CLinkedListNode<T>* Last();
```

### Возвращаемое значение

Возвращает указатель на последний узел.

## Contains

Определяет, содержит ли двунаправленный список элемент с указанным значением.

```
bool Contains(  
    T   item      // искомое значение  
);
```

### Параметры

*item*

[in] Искомое значение.

### Возвращаемое значение

Возвращает true, если в двунаправленном списке есть элемент с указанным значением, иначе false.

## CopyTo

Копирует все элементы двунаправленного списка в указанный массив, начиная с определенного индекса.

```
int CopyTo(  
    T&           dst_array[],      // массив для записи  
    const int    dst_start=0        // начальный индекс для записи  
);
```

### Параметры

*&dst\_array[]*

[out] Массив, в который будут записаны элементы двунаправленного списка.

*dst\_start=0*

[in] Индекс в массиве, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных элементов.

## Clear

Удаляет все элементы коллекции.

```
void Clear();
```

## Remove

Удаляет первое вхождение указанного элемента из двунаправленного списка.

Версия для удаления элемента по значению.

```
bool Remove(  
    T item  
);  
// значение элемента
```

Версия для удаления элемента по указателю на узел.

```
bool Remove(  
    CLinkedListNode<T>* node  
);  
// узел элемента
```

### Параметры

*item*  
[in] Значение элемента, которое нужно удалить.  
*\*node*  
[in] Узел элемента, который нужно удалить.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## RemoveFirst

Удаляет первый элемент из двунаправленного списка.

```
bool RemoveFirst();
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## RemoveLast

Удаляет последний элемент из двунаправленного списка.

```
bool RemoveLast();
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Find

Ищет первое вхождение заданного значения в двунаправленный список.

```
CLinkedListNode<T>* Find(
    T   value      // искомое значение
);
```

### Параметры

*value*

[in] Искомое значение.

### Возвращаемое значение

Возвращает указатель на первый найденный узел, содержащий искомое значение в случае успеха, иначе NULL.

## FindLast

Ищет последнее вхождение заданного значения в двунаправленный список.

```
CLinkedListNode<T>* FindLast (
    T   value      // искомое значение
);
```

### Параметры

*value*

[in] Искомое значение.

### Возвращаемое значение

Возвращает указатель на последний найденный узел, содержащий искомое значение, в случае успеха, иначе NULL.

## CQueue<T>

Класс CQueue<T> – шаблонный класс, реализующий интерфейс ICollection<T>.

### Описание

Класс CQueue<T> – динамическая коллекция данных типа T, организованная в виде очереди, который работает по принципу "первым вошёл – первым вышел" (FIFO).

### Декларация

```
template<typename T>
class CQueue : public ICollection<T>
```

### Заголовок

```
#include <Generic\Queue.mqh>
```

### Иерархия наследования

[ICollection](#)

CQueue

### Методы класса

Метод	Описание
<a href="#">Add</a>	Добавляет элемент в очередь
<a href="#">Enqueue</a>	Добавляет элемент в очередь
<a href="#">Count</a>	Возвращает количество элементов в очереди
<a href="#">Contains</a>	Определяет, содержит ли очередь элемент с указанным значением
<a href="#">TrimExcess</a>	Сокращает емкость очереди до фактического числа элементов, тем самым освобождая неиспользованную память
<a href="#">CopyTo</a>	Копирует все элементы очереди в указанный массив, начиная с определенного индекса
<a href="#">Clear</a>	Удаляет все элементы очереди
<a href="#">Remove</a>	Удаляет первое вхождение указанного элемента из очереди
<a href="#">Dequeue</a>	Возвращает начальный элемент, удаляя его из очереди
<a href="#">Peek</a>	Возвращает начальный элемент, не удаляя его из очереди

## Add

Добавляет элемент в очередь.

```
bool Add(  
    T   value      // значение элемента  
);
```

### Параметры

*value*

[in] Значение элемента для добавления.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## .Enqueue

Добавляет элемент в очередь.

```
bool Enqueue(
    T   value        // элемент для добавления
);
```

### Параметры

*value*

[in] Элемент для добавления.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Count

Возвращает количество элементов в очереди.

```
int Count();
```

### Возвращаемое значение

Возвращает количество элементов.

## Contains

Определяет, содержит ли очередь элемент с указанным значением.

```
bool Contains(  
    T   item      // искомое значение  
) ;
```

### Параметры

*item*

[in] Искомое значение.

### Возвращаемое значение

Возвращает true, если в очереди есть элемент с указанным значением, иначе false.

## TrimExcess

Сокращает емкость очереди до фактического числа элементов, тем самым освобождая неиспользованную память.

```
void TrimExcess();
```

## CopyTo

Копирует все элементы очереди в указанный массив, начиная с определенного индекса.

```
int CopyTo(
    T&          dst_array[],      // массив для записи
    const int    dst_start=0       // начальный индекс для записи
);
```

### Параметры

`&dst_array[]`

[out] Массив, в который будут записаны элементы очереди.

`dst_start=0`

[in] Индекс в массиве, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных элементов.

## Clear

Удаляет все элементы очереди.

```
void Clear();
```

## Remove

Удаляет первое вхождение указанного элемента из очереди.

```
bool Remove(  
    T    item      // значение элемента  
);
```

### Параметры

*item*

[in] Значение элемента, которое нужно удалить.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Dequeue

Возвращает начальный элемент, удаляя его из очереди.

```
T Dequeue();
```

### Возвращаемое значение

Возвращает начальный элемент.

## Peek

Возвращает начальный элемент, не удаляя его из очереди.

```
T Peek();
```

### Возвращаемое значение

Возвращает начальный элемент.

## CRedBlackTree<T>

Класс CRedBlackTree<T> – шаблонный класс, реализующий интерфейс ICollection<T>.

### Описание

Класс CRedBlackTree<T> – реализация динамического красно-черного дерева, в узлах которого хранятся данные типа T. Класс предоставляет основные методы для работы с красно-черными деревьями: добавление, удаление, поиск максимального, минимального значения и другие.

### Декларация

```
template<typename T>
class CRedBlackTree : public ICollection<T>
```

### Заголовок

```
#include <Generic\RedBlackTree.mqh>
```

### Иерархия наследования

[ICollection](#)

CRedBlackTree

### Методы класса

Метод	Описание
<a href="#">Add</a>	Добавляет элемент в красно-черное дерево
<a href="#">Root</a>	Возвращает указатель на корень красно-черного дерева
<a href="#">Count</a>	Возвращает количество элементов в красно-черном дереве
<a href="#">Contains</a>	Определяет, содержит ли красно-черное дерево элемент с указанным значением
<a href="#">Comparer</a>	Возвращает указатель на интерфейс IComparer<T>, использующийся для организации красно-черного дерева
<a href="#">TryGetMin</a>	Получает минимальный элемент из красно-черного дерева
<a href="#">TryGetMax</a>	Получает максимальный элемент из красно-черного дерева
<a href="#">CopyTo</a>	Копирует все элементы красно-черного дерева в указанный массив, начиная с определенного индекса
<a href="#">Clear</a>	Удаляет все элементы красно-черного дерева

<a href="#"><u>Remove</u></a>	Удаляет вхождение указанного элемента из красно-черного дерева
<a href="#"><u>RemoveMin</u></a>	Удаляет элемент с минимальным значением из красно-черного дерева
<a href="#"><u>RemoveMax</u></a>	Удаляет элемент с максимальным значением из красно-черного дерева
<a href="#"><u>Find</u></a>	Ищет вхождение заданного значения в красно-черное дерево
<a href="#"><u>FindMax</u></a>	Ищет элемент с максимальным значением в красно-черное дерево
<a href="#"><u>FindMin</u></a>	Ищет элемент с минимальным значением в красно-черном дереве

## Add

Добавляет элемент в красно-черное дерево.

```
bool Add(  
    T   value      // элемент для добавления  
);
```

### Параметры

*value*

[in] Элемент для добавления.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Count

Возвращает количество элементов в красно-черном дереве.

```
int Count();
```

### Возвращаемое значение

Возвращает количество элементов.

## Root

Возвращает указатель на корень красно-черного дерева.

```
CRedBlackTreeNode<T>* Root();
```

### Возвращаемое значение

Возвращает указатель на корень.

## Contains

Определяет, содержит ли красно-черное дерево элемент с указанным значением.

```
bool Contains(  
    T   item      // искомое значение  
) ;
```

### Параметры

*item*

[in] Искомое значение.

### Возвращаемое значение

Возвращает true если в красно-черном дереве есть элемент с указанным значением, иначе false.

## Comparer

Возвращает указатель на интерфейс `IComparer<T>`, использующийся для организации красно-черного дерева.

```
IComparer<T>* Comparer() const;
```

### Возвращаемое значение

Возвращает указатель на интерфейс `IComparer<T>`.

## TryGetMin

Получает минимальный элемент из красно-черного дерева.

```
bool TryGetMin(  
    T& min          // переменная для записи значения  
);
```

### Параметры

*&min*

[out] Переменная, в которую будет записано минимальное значение.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## TryGetMax

Получает максимальный элемент из красно-черного дерева.

```
bool TryGetMax(  
    T& max           // переменная для записи значения  
);
```

### Параметры

*&max*

[out] Переменная, в которую будет записано максимальное значение.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## CopyTo

Копирует все элементы красно-черного дерева в указанный массив, начиная с определенного индекса.

```
int CopyTo(  
    T&           dst_array[],      // массив для записи  
    const int    dst_start=0        // начальный индекс для записи  
);
```

### Параметры

*&dst\_array[]*

[out] Массив, в который будут записаны элементы красно-черного дерева.

*dst\_start=0*

[in] Индекс в массиве, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных элементов.

## Clear

Удаляет все элементы красно-черного дерева.

```
void Clear();
```

## Remove

Удаляет вхождение указанного элемента из красно-черного дерева.

**Версия для удаления элемента по заданному значению.**

```
bool Remove(  
    T value // значение элемента  
) ;
```

**Версия для удаления элемента по указателю на узел.**

```
bool Remove(  
    CRedBlackTreeNode<T>* node // узел элемента  
) ;
```

### Параметры

*item*

[in] Значение элемента, которое нужно удалить.

*\*node*

[in] Узел элемента, который нужно удалить.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## RemoveMin

Удаляет элемент с минимальным значением из красно-черного дерева.

```
bool RemoveMin();
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## RemoveMax

Удаляет элемент с максимальным значением из красно-черного дерева.

```
bool RemoveMax();
```

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Find

Ищет вхождение заданного значения в красно-черное дерево.

```
CRedBlackTreeNode<T>* Find(
    T   value      // искомое значение
);
```

### Параметры

*value*

[in] Искомое значение.

### Возвращаемое значение

Возвращает указатель на найденный узел, содержащий искомое значение в случае успеха, иначе NULL.

## FindMin

Ищет элемент с минимальным значением в красно-черном дереве.

```
CRedBlackTreeNode<T>* FindMin();
```

### Возвращаемое значение

Возвращает указатель на узел, содержащий минимальное значение в случае успеха, иначе NULL.

## FindMax

Ищет элемент с максимальным значением в красно-черное дерево.

```
CRedBlackTreeNode<T>* FindMax();
```

### Возвращаемое значение

Возвращает указатель на узел, содержащий максимальное значение в случае успеха, иначе NULL.

## CSortedMap< TKey, TValue >

Класс CSortedMap< TKey, TValue > – шаблонный класс, реализующий интерфейс IMap< TKey, TValue >.

### Описание

Класс CSortedMap< TKey, TValue > – реализация динамической хэш-таблицы, данные которой хранятся в виде пар "ключ – значение", отсортированных по ключу и с соблюдением требования уникальности ключа. Данный класс предоставляет основные методы для работы с хэш-таблицей: доступ к значению по ключу, поиск и удаление пары "ключ-значение" и другие.

### Декларация

```
template<typename TKey, typename TValue>
class CSortedMap : public IMap<TKey, TValue>
```

### Заголовок

```
#include <Generic\SortedMap.mqh>
```

### Иерархия наследования

[ICollection](#)

[IMap](#)

CSortedMap

### Методы класса

Метод	Описание
<a href="#"><u>Add</u></a>	Добавляет пару "ключ – значение" в хэш-таблицу
<a href="#"><u>Count</u></a>	Возвращает количество элементов в отсортированной хэш-таблице
<a href="#"><u>Contains</u></a>	Определяет, содержит ли сортированная хэш-таблица указанную пару "ключ – значение"
<a href="#"><u>ContainsKey</u></a>	Определяет, содержит ли сортированная хэш-таблица пару "ключ – значение" с указанным ключом
<a href="#"><u>ContainsValue</u></a>	Определяет, содержит ли сортированная хэш-таблица пару "ключ – значение" с указанным значением
<a href="#"><u>Comparer</u></a>	Возвращает указатель на интерфейс IComparer<T>, использующийся для организации сортированной хэш-таблицы
<a href="#"><u>CopyTo</u></a>	Копирует все пары "ключ – значение" из сортированной хэш-таблицы в указанные

	массивы, начиная с определенного индекса
<a href="#"><u>Clear</u></a>	Удаляет все элементы сортированной хэш-таблицы
<a href="#"><u>Remove</u></a>	Удаляет первое вхождение пары "ключ — значение" сортированной хэш-таблицы
<a href="#"><u>TryGetValue</u></a>	Получает элемент из сортированной хэш-таблицы по заданному ключу
<a href="#"><u>TrySetValue</u></a>	Изменяет значение пары "ключ — значение" из отсортированной хэш-таблицы по заданному ключу

## Add

Добавляет пару "ключ – значение" в хэш-таблицу.

Версия для добавления сформированной пары "ключ – значение".

```
bool Add(  
    CKeyValuePair< TKey TValue>* pair // пара "ключ – значение"  
) ;
```

Версия для добавления новой пары "ключ – значение" с указанным ключом и значением.

```
bool Add(  
    TKey key, // ключ  
    TValue value // значение  
) ;
```

### Параметры

*\*pair*

[in] Пары "ключ – значение".

*key*

[in] Ключ.

*value*

[in] Значение.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Count

Возвращает количество элементов в отсортированной хэш-таблице.

```
int Count();
```

## Comparer

Возвращает указатель на интерфейс `IComparer<T>`, использующийся для организации сортированной хэш-таблицы.

```
IComparer< TKey>* Comparer() const;
```

### Возвращаемое значение

Возвращает указатель на интерфейс `IComparer<T>`.

## Contains

Определяет, содержит ли сортированная хэш-таблица указанную пару "ключ — значение".

**Версия для работы с сформированной парой "ключ — значение".**

```
bool Contains(
    CKeyValuePair< TKey TValue>* item      // пара "ключ — значение"
);
```

**Версия для работы с парой "ключ — значение" в виде отдельно заданного ключа и значения.**

```
bool Contains(
    TKey key,                      // ключ
    TValue value                   // значение
);
```

### Параметры

*\*item*  
[in] Пара "ключ — значение".

*key*  
[in] Ключ.

*value*  
[in] Значение.

### Возвращаемое значение

Возвращает true, если в сортированной хэш-таблице есть пара "ключ — значение" с указанным ключом и значением, иначе false.

## ContainsKey

Определяет, содержит ли сортированная хэш-таблица пару "ключ — значение" с указанным ключом.

```
bool ContainsKey(  
    TKey key // ключ  
) ;
```

### Параметры

*key*  
[in] Ключ.

### Возвращаемое значение

Возвращает true, если в сортированной хэш-таблице есть пара "ключ — значение" с указанным ключом, иначе false.

## ContainsValue

Определяет, содержит ли сортированная хэш-таблица пару "ключ — значение" с указанным значением.

```
bool ContainsValue(  
    TValue  value      // значение  
);
```

### Параметры

*value*  
[in] Значение.

### Возвращаемое значение

Возвращает true, если в сортированной хэш-таблице есть пара "ключ — значение" с указанным значением, иначе false.

## CopyTo

Копирует все пары "ключ – значение" из сортированной хэш-таблицы в указанные массивы, начиная с определенного индекса.

**Версия для копирования сортированной хэш-таблицы в массив пар "ключ-значение".**

```
int CopyTo(
    CKeyValuePair< TKey TValue>*& dst_array[],           // массив для записи пар "ключ-значение"
    const int                      dst_start=0            // начальный индекс для записи
);
```

**Версия для копирования сортированной хэш-таблицы в отдельные массивы для ключей и значений.**

```
int CopyTo(
    TKey&      dst_keys[],                // массив для записи ключей
    TValue&    dst_values[],              // массив для записи значений
    const int   dst_start=0               // начальный индекс для записи
);
```

### Параметры

`*&dst_array[]`

[out] Массив, в который будут записаны все пары из хэш-таблицы.

`&dst_keys[]`

[out] Массив, в который будут записаны все ключи из хэш-таблицы.

`&dst_values[]`

[out] Массив, в который будут записаны все значения из хэш-таблицы.

`dst_start=0`

[in] Индекс в массивах, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных пар "ключ-значение".

## Clear

Удаляет все элементы сортированной хэш-таблицы.

```
void Clear();
```

## Remove

Удаляет первое вхождение пары "ключ – значение" сортированной хэш-таблицы.

Версия для удаления пары "ключ – значение" по сформированной паре "ключ – значение".

```
bool Remove(  
    CKeyValuePair< TKey TValue>*& item // пара "ключ – значение"  
) ;
```

Версия для удаления пары "ключ – значение" по ключу.

```
bool Remove(  
    TKey key // ключ  
) ;
```

### Параметры

\*item

[in] Пары "ключ – значение".

key

[in] Ключ.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## TryGetValue

Получает элемент из сортированной хэш-таблицы по заданному ключу.

```
bool TryGetValue(  
    TKey      key,          // ключ  
    TValue&   value        // переменная для записи значения  
);
```

### Параметры

*key*

[in] Ключ.

*&value*

[out] Переменная, в которую будет записано указанное значение пары "ключ — значение".

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## TrySetValue

Изменяет значение пары "ключ — значение" из сортированной хэш-таблицы по заданному ключу.

```
bool TrySetValue(  
    TKey     key,      // Ключ  
    TValue   value     // новое значение  
);
```

### Параметры

*key*

[in] Ключ.

*value*

[in] Новое значение, которое нужно присвоить указанной паре "ключ — значение".

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## CSortedSet<T>

Класс CSortedSet<T> — шаблонный класс, реализующий интерфейс ISet<T>.

### Описание

Класс CSortedSet<T> является реализацией отсортированного динамического множества данных типа T, с соблюдением требования уникальности каждого значения. Данный класс предоставляет основные методы для работы с множествами и операциями над ними: объединение и пересечение множеств, определение строгих и нестрогих подмножеств и другие.

### Декларация

```
template<typename T>
class CSortedSet : public ISet<T>
```

### Заголовок

```
#include <Generic\SortedSet.mqh>
```

### Иерархия наследования

[ICollection](#)

[ISet](#)

CSortedSet

### Методы класса

Метод	Описание
<a href="#">Add</a>	Добавляет элемент в сортированное множество
<a href="#">Count</a>	Возвращает количество элементов в сортированном множестве
<a href="#">Contains</a>	Определяет, содержит ли сортированное множество элемент с указанным значением
<a href="#">Comparer</a>	Возвращает указатель на интерфейс IComparer<T>, использующийся для организации сортированного множества
<a href="#">TryGetMin</a>	Получает минимальный элемент из сортированного множества
<a href="#">TryGetMax</a>	Получает максимальный элемент из сортированного множества
<a href="#">CopyTo</a>	Копирует все элементы сортированного множества в указанный массив, начиная с определенного индекса

<a href="#"><u>Clear</u></a>	Удаляет все элементы сортированного множества
<a href="#"><u>Remove</u></a>	Удаляет вхождение указанного элемента из сортированного множества
<a href="#"><u>ExceptWith</u></a>	Выполняет операцию разности текущей и переданной коллекции (массива)
<a href="#"><u>IntersectWith</u></a>	Выполняет операцию пересечения текущей и переданной коллекции (массива)
<a href="#"><u>SymmetricExceptWith</u></a>	Выполняет операцию симметричной разности текущей и переданной коллекции (массива)
<a href="#"><u>UnionWith</u></a>	Выполняет операцию объединения текущей и переданной коллекции (массива)
<a href="#"><u>IsProperSubsetOf</u></a>	Определяет, является ли текущее сортированное множество строгим подмножеством заданной коллекции или массива
<a href="#"><u>IsProperSupersetOf</u></a>	Определяет, является ли текущее сортированное множество строгим надмножеством заданной коллекции или массива
<a href="#"><u>IsSubsetOf</u></a>	Определяет, является ли текущее сортированное множество подмножеством заданной коллекции или массива
<a href="#"><u>IsSupersetOf</u></a>	Определяет, является ли текущее сортированное множество надмножеством заданной коллекции или массива
<a href="#"><u>Overlaps</u></a>	Определяет, пересекается ли текущее сортированное множество с заданной коллекцией или массивом
<a href="#"><u>SetEquals</u></a>	Определяет, содержит ли текущее сортированное множество все элементы из заданной коллекции или массива
<a href="#"><u>GetViewBetween</u></a>	Получает от текущего сортированного множества подмножество, заданное минимальным и максимальным значением
<a href="#"><u>GetReverse</u></a>	Получает копию текущего сортированного множества, где все элементы расположены в обратном порядке

## Add

Добавляет элемент в сортированное множество.

```
bool Add(  
    T   value      // значение элемента  
);
```

### Параметры

*value*

[in] Значение элемента для добавления.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Count

Возвращает количество элементов в сортированном множестве.

```
int Count();
```

### Возвращаемое значение

Возвращает количество элементов.

## Contains

Определяет, содержит ли сортированное множество элемент с указанным значением.

```
bool Contains(  
    T   item      // искомое значение  
);
```

### Параметры

*item*

[in] Искомое значение.

### Возвращаемое значение

Возвращает true, если во множестве есть элемент с указанным значением, иначе false.

## Comparer

Возвращает указатель на интерфейс `IComparer<T>`, использующийся для организации сортированного множества.

```
IComparer<T>* Comparer() const;
```

### Возвращаемое значение

Возвращает указатель на интерфейс `IComparer<T>`.

## TryGetMin

Получает минимальный элемент из сортированного множества.

```
bool TryGetMin(  
    T& min // переменная для записи значения  
) ;
```

### Параметры

*&min*

[out] Переменная, в которую будет записано минимальное значение.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## TryGetMax

Получает максимальный элемент из сортированного множества.

```
bool TryGetMax(  
    T& max           // переменная для записи значения  
);
```

### Параметры

*&max*

[out] Переменная, в которую будет записано максимальное значение.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## CopyTo

Копирует все элементы сортированного множества в указанный массив, начиная с определенного индекса.

```
int CopyTo(  
    T&           dst_array[],      // массив для записи  
    const int    dst_start=0        // начальный индекс для записи  
);
```

### Параметры

*&dst\_array[]*  
[out] Массив, в который будут записаны элементы множества.  
*dst\_start=0*  
[in] Индекс в массиве, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных элементов.

## Clear

Удаляет все элементы сортированного множества.

```
void Clear();
```

## Remove

Удаляет вхождение указанного элемента из сортированного множества.

```
bool Remove(  
    T    item      // значение элемента  
);
```

### Параметры

*item*

[in] Значение элемента, которое нужно удалить.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## ExceptWith

Выполняет операцию разности текущей и переданной коллекции (массива). То есть удаляет из текущей коллекции (массива) все элементы, которые есть также и в указанной коллекции (массиве).

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void ExceptWith(
    ICollection<T>* collection      // коллекция
);
```

Версия для работы с массивом.

```
void ExceptWith(
    T& array[]                  // массив
);
```

### Параметры

`*collection`

[in] Коллекция, которая будет вычитаться из текущего сортированного множества.

`&collection[]`

[in] Массив, который будет вычитаться из текущего сортированного множества.

### Примечание

Результат записывается в текущую коллекцию (массив).

## IntersectWith

Выполняет операцию пересечения текущей и переданной коллекции (массива). То есть оставляет в текущей коллекции только те элементы, которые имеются также и в указанной коллекции (массиве).

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void IntersectWith(
    ICollection<T>* collection      // коллекция
);
```

Версия для работы с массивом.

```
void IntersectWith(
    T& array[]                      // массив
);
```

### Параметры

`*collection`

[in] Коллекция, с которой будет строиться произведение.

`&collection[]`

[in] Массив, с которым будет строиться произведение.

### Примечание

Результат записывается в текущую коллекцию (массив).

## SymmetricExceptWith

Выполняет операцию симметричной разности текущей и переданной коллекции (массива). То есть в текущей коллекции (массиве) будут содержаться только те элементы, которые имелись либо в исходном объекте, либо в переданном, но не одновременно в них обоих.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void SymmetricExceptWith(
    ICollection<T>* collection      // коллекция
);
```

Версия для работы с массивом.

```
void SymmetricExceptWith(
    T& array[]                      // массив
);
```

### Параметры

`*collection`

[in] Коллекция, с которой будет строиться симметричная разность.

`&collection[]`

[in] Массив, с которым будет строиться симметричная разность.

### Примечание

Результат записывается в текущую коллекцию (массив).

## UnionWith

Выполняет операцию объединения текущей и переданной коллекции (массива). То есть добавляет в текущую коллекцию (массив) отсутствующие элементы из указанной коллекции (массива).

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
void UnionWith(  
    ICollection<T>* collection      // коллекция  
) ;
```

Версия для работы с массивом.

```
void UnionWith(  
    T& array[]                      // массив  
) ;
```

### Параметры

`*collection`

[in] Коллекция, с которой будет строиться сумма.

`&collection[]`

[in] Массив, с которым будет строиться сумма.

### Примечание

Результат записывается в текущую коллекцию (массив).

## IsProperSubsetOf

Определяет, является ли текущее сортированное множество строгим подмножеством заданной коллекции или массива.

Версия для работы с коллекцией реализующей интерфейс `ICollection<T>`.

```
bool IsProperSubsetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

Версия для работы с массивом.

```
bool IsProperSubsetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true`, если текущее сортированное множество является строгим подмножеством, иначе `false`.

## IsProperSupersetOf

Определяет, является ли текущее сортированное множество строгим надмножеством заданной коллекции или массива.

**Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.**

```
bool IsProperSupersetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

**Версия для работы с массивом.**

```
bool IsProperSupersetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

*\*collection*

[in] Коллекция для определения отношения.

*&collection[]*

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true` в случае, если текущее сортированное множество является строгим надмножеством, иначе `false`.

## IsSubsetOf

Определяет, является ли текущее сортированное множество подмножеством заданной коллекции или массива.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool IsSubsetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

Версия для работы с массивом.

```
bool IsSubsetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true` в случае, если текущее сортированное множество является подмножеством, иначе `false`.

## IsSupersetOf

Определяет, является ли текущее сортированное множество надмножеством заданной коллекции или массива.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool IsSupersetOf(
    ICollection<T>* collection      // коллекция для определения отношения
);
```

Версия для работы с массивом.

```
bool IsSupersetOf(
    T& array[]                  // массив для определения отношения
);
```

### Параметры

`*collection`

[in] Коллекция для определения отношения.

`&collection[]`

[in] Массив для определения отношения.

### Возвращаемое значение

Возвращает `true` в случае, если текущее сортированное множество является надмножеством, иначе `false`.

## Overlaps

Определяет, пересекается ли текущее сортированное множество с заданной коллекцией или массивом.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool Overlaps(
    ICollection<T>* collection      // коллекция для сравнения
);
```

Версия для работы с массивом.

```
bool Overlaps(
    T& array[]                  // массив для сравнения
);
```

### Параметры

`*collection`

[in] Коллекция для определения пересечения.

`&collection[]`

[in] Массив для определения пересечения.

### Возвращаемое значение

Возвращает `true` в случае, если между текущим сортированным множеством и коллекцией или массивом есть пересечение, иначе `false`.

## SetEquals

Определяет, содержит ли текущее сортированное множество все элементы из заданной коллекции или массива.

Версия для работы с коллекцией, реализующей интерфейс `ICollection<T>`.

```
bool SetEquals(
    ICollection<T>* collection      // коллекция для сравнения
);
```

Версия для работы с массивом.

```
bool SetEquals(
    T& array[]                  // массив для сравнения
);
```

### Параметры

`*collection`

[in] Коллекция для сопоставления элементов.

`&collection[]`

[in] Массив для сопоставления элементов.

### Возвращаемое значение

Возвращает `true` в случае, если в текущем сортированном множестве есть все элементы указанной коллекции или массива, иначе `false`.

## GetViewBetween

Получает от текущего сортированного множества подмножество, заданное минимальным и максимальным значением.

```
bool GetViewBetween(
    T& array[],           // массив для записи
    T lower_value,        // минимальное значение
    T upper_value         // максимальное значение
);
```

### Параметры

*&array[]*

[out] Массив для записи подмножества.

*lower\_value*

[in] Минимальное значение диапазона.

*upper\_value*

[in] Максимальное значение диапазона.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## GetReverse

Получает копию текущего сортированного множества, где все элементы расположены в обратном порядке.

```
bool GetReverse(
    T& array[]          // массив для записи
);
```

### Параметры

*&array[]*  
[out] Массив для записи.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## CStack<T>

Класс CStack<T> – шаблонный класс, реализующий интерфейс ICollection<T>.

### Описание

Класс CStack<T> – динамическая коллекция данных типа T, организованная в виде стека, который работает по принципу "последним пришел – первым вышел" (LIFO).

### Декларация

```
template<typename T>
class CStack : public ICollection<T>
```

### Заголовок

```
#include <Generic\Stack.mqh>
```

### Иерархия наследования

[ICollection](#)

CStack

### Методы класса

Метод	Описание
<a href="#">Add</a>	Добавляет элемент в стек
<a href="#">Count</a>	Возвращает количество элементов в стеке
<a href="#">Contains</a>	Определяет, содержит ли стек элемент с указанным значением
<a href="#">TrimExcess</a>	Сокращает емкость стека до фактического числа элементов
<a href="#">CopyTo</a>	Копирует все элементы стека в указанный массив, начиная с определенного индекса
<a href="#">Clear</a>	Удаляет все элементы стека
<a href="#">Remove</a>	Удаляет первое вхождение указанного элемента из стека
<a href="#">Push</a>	Добавляет элемент в стек
<a href="#">Peek</a>	Возвращает головной элемент, не удаляя его из стека
<a href="#">Pop</a>	Возвращает головной элемент, удаляя его из стека

## Add

Добавляет элемент в стек.

```
bool Add(  
    T   value      // значение элемента  
);
```

### Параметры

*value*

[in] Значение элемента для добавления.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Count

Возвращает количество элементов в стеке.

```
int Count();
```

### Возвращаемое значение

Возвращает количество элементов.

## Contains

Определяет, содержит ли стек элемент с указанным значением.

```
bool Contains(  
    T   item      // искомое значение  
);
```

### Параметры

*item*

[in] Искомое значение.

### Возвращаемое значение

Возвращает true, если в стеке есть элемент с указанным значением, иначе false.

## TrimExcess

Сокращает емкость стека до фактического числа элементов, тем самым освобождая неиспользуемую память.

```
void TrimExcess();
```

## CopyTo

Копирует все элементы стека в указанный массив, начиная с определенного индекса.

```
int CopyTo(
    T&          dst_array[],      // массив для записи
    const int    dst_start=0       // начальный индекс для записи
);
```

### Параметры

`&dst_array[]`

[out] Массив, в который будут записаны элементы стека.

`dst_start=0`

[in] Индекс в массиве, с которого начинается копирование.

### Возвращаемое значение

Возвращает количество скопированных элементов.

## Clear

Удаляет все элементы стека.

```
void Clear();
```

## Remove

Удаляет первое вхождение указанного элемента из стека.

```
bool Remove(  
    T    item      // значение элемента  
);
```

### Параметры

*item*

[in] Значение элемента, которое нужно удалить.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Push

Добавляет элемент в стек.

```
bool Push(  
    Т   value      // элемент для добавления  
);
```

### Параметры

*value*

[in] Элемент для добавления.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Peek

Возвращает головной элемент, не удаляя его из стека.

```
T Peek();
```

### Возвращаемое значение

Возвращает головной элемент.

## Pop

Возвращает головной элемент, удаляя его из стека.

```
T Pop();
```

### Возвращаемое значение

Возвращает головной элемент.

## ArrayBinarySearch

Ищет указанное значение в отсортированном по возрастанию одномерном массиве, используя интерфейс `IComparable<T>` для сравнения элементов.

```
template<typename T>
int ArrayBinarySearch(
    T&           array[],          // массив для поиска
    const int      start_index,     // начальный индекс
    const int      count,           // диапазон поиска
    T             value,            // искомое значение
    IComparer<T>* comparer        // интерфейс для сравнения
);
```

### Параметры

`&array[]`

[out] Массив для поиска.

`value`

[in] Искомое значение.

`*comparer`

[in] Интерфейс для сравнения элементов.

`start_index`

[in] Начальный индекс, с которого начинается поиск.

`count`

[in] Длина диапазона поиска.

### Возвращаемое значение

Возвращает индекс найденного элемента. Если искомое значение не найдено, то возвращает индекс меньшего элемента, которое ближе всех по значению.

## ArrayIndexOf

Ищет первое вхождение значения в одномерном массиве.

```
template<typename T>
int ArrayIndexOf(
    T&           array[],          // массив для поиска
    T            value,             // искомое значение
    const int    start_index,      // начальный индекс
    const int    count              // диапазон поиска
);
```

### Параметры

*&array[]*

[out] Массив для поиска.

*value*

[in] Искомое значение.

*start\_index*

[in] Начальный индекс, с которого начинается поиск.

*count*

[in] Длина диапазона поиска.

### Возвращаемое значение

Возвращает индекс первого найденного элемента. Если значение не найдено, возвращает -1.

## ArrayLastIndexOf

Ищет последнее вхождение значения в одномерном массиве.

```
template<typename T>
int ArrayLastIndexOf(
    T&           array[],          // массив для поиска
    T             value,            // искомое значение
    const int     start_index,      // начальный индекс
    const int     count            // диапазон поиска
);
```

### Параметры

*&array[]*

[out] Массив для поиска.

*value*

[in] Искомое значение.

*start\_index*

[in] Начальный индекс, с которого начинается поиск.

*count*

[in] Длина диапазона поиска.

### Возвращаемое значение

Возвращает индекс последнего найденного элемента. Если значение не найдено, возвращает -1.

## ArrayReverse

Изменяет последовательность элементов в одномерном массиве.

```
template<typename T>
bool ArrayReverse(
    T&           array[],          // исходный массив
    const int     start_index,      // начальный индекс
    const int     count            // количество элементов
);
```

### Параметры

*array[]*

[out] Исходный массив.

*start\_index*

[in] Начальный индекс.

*count*

[in] Количество элементов массива, участвующих в операции.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## Compare

Сравнивает два значения на отношение "Больше, меньше или равно".

**Версия для сравнения двух значений типа bool.**

```
int Compare(
    const bool x,           // первое значение
    const bool y            // второе значение
);
```

**Версия для сравнения двух значений типа char.**

```
int Compare(
    const char x,           // первое значение
    const char y            // второе значение
);
```

**Версия для сравнения двух значений типа uchar.**

```
int Compare(
    const uchar x,          // первое значение
    const uchar y            // второе значение
);
```

**Версия для сравнения двух значений типа short.**

```
int Compare(
    const short x,           // первое значение
    const short y            // второе значение
);
```

**Версия для сравнения двух значений типа ushort.**

```
int Compare(
    const ushort x,          // первое значение
    const ushort y            // второе значение
);
```

**Версия для сравнения двух значений типа color.**

```
int Compare(
    const color x,           // первое значение
    const color y            // второе значение
);
```

**Версия для сравнения двух значений типа int.**

```
int Compare(
    const int x,              // первое значение
    const int y                // второе значение
);
```

**Версия для сравнения двух значений типа uint.**

```
int Compare(
    const uint  x,           // первое значение
    const uint  y            // второе значение
);
```

**Версия для сравнения двух значений типа datetime.**

```
int Compare(
    const datetime  x,       // первое значение
    const datetime  y        // второе значение
);
```

**Версия для сравнения двух значений типа long.**

```
int Compare(
    const long  x,          // первое значение
    const long  y            // второе значение
);
```

**Версия для сравнения двух значений типа ulong.**

```
int Compare(
    const ulong  x,         // первое значение
    const ulong  y           // второе значение
);
```

**Версия для сравнения двух значений типа float.**

```
int Compare(
    const float  x,         // первое значение
    const float  y           // второе значение
);
```

**Версия для сравнения двух значений типа double.**

```
int Compare(
    const double  x,         // первое значение
    const double  y           // второе значение
);
```

**Версия для сравнения двух значений типа string.**

```
int Compare(
    const string  x,         // первое значение
    const string  y           // второе значение
);
```

**Версия для сравнения двух значений остальных типов.**

```
template<typename T>
int Compare(
```

```
T x, // первое значение  
T y // второе значение  
);
```

## Параметры

*x*

[in] Первое значение.

*y*

[in] Второе значение

## Возвращаемое значение

Возвращает число, выражающее отношение двух сравниемых значений:

- результат меньше нуля — *x* меньше *y* (*x*<*y*)
- результат равен нулю — *x* равен *y* (*x*=*y*)
- результат больше нуля — *x* больше *y* (*x*>*y*)

## Примечание

Если тип *T* — объект, реализующий интерфейс *IComparable<T>*, то объекты будут сравниваться на основе его метода сравнения *Compare*. Во всех остальных случаях возвращается 0.

## Equals

Сравнивает два значения на равенство.

```
template<typename T>
bool Equals(
    T x,           // первое значение
    T y            // второе значение
);
```

### Параметры

*x*

[in] Первое значение.

*y*

[in] Второе значение.

### Возвращаемое значение

Возвращает true, если объекты равны, иначе false.

### Примечание

Если тип *T* – объект, реализующий интерфейс *IEqualityComparable<T>*, то объекты будут сравниваться на основе его метода сравнения *Equals*. Во всех остальных случаях применяется стандартное сравнение на равенство.

## GetHashCode

Вычисляет значение хэш-кода.

**Версия для работы с типом bool.**

```
int GetHashCode(
    const bool  value          // значение
);
```

**Версия для работы с типом char.**

```
int GetHashCode(
    const char   value          // значение
);
```

**Версия для работы с типом uchar.**

```
int GetHashCode(
    const uchar  value          // значение
);
```

**Версия для работы с типом short.**

```
int GetHashCode(
    const short  value          // значение
);
```

**Версия для работы с типом ushort.**

```
int GetHashCode(
    const ushort  value          // значение
);
```

**Версия для работы с типом color.**

```
int GetHashCode(
    const color   value          // значение
);
```

**Версия для работы с типом int.**

```
int GetHashCode(
    const int    value          // значение
);
```

**Версия для работы с типом uint.**

```
int GetHashCode(
    const uint   value          // значение
);
```

**Версия для работы с типом datetime.**

```
int GetHashCode(
```

```
    const datetime value      // значение
);
```

**Версия для работы с типом long.**

```
int GetHashCode(
    const long value        // значение
);
```

**Версия для работы с типом ulong.**

```
int GetHashCode(
    const ulong value       // значение
);
```

**Версия для работы с типом float.**

```
int GetHashCode(
    const float value       // значение
);
```

**Версия для работы с типом double.**

```
int GetHashCode(
    const double value      // значение
);
```

**Версия для работы с типом string.**

```
int GetHashCode(
    const string value      // значение
);
```

**Версия для работы с остальными типами.**

```
template<typename T>
int GetHashCode(
    T value                  // значение
);
```

## Параметры

**value**

[in] Значение, для которого нужно получить хэш-код.

## Возвращаемое значение

Возвращает хэш-код.

## Примечание

Если тип T – объект, реализующий интерфейс IEqualityComparer<T>, то хэш-код будет получен на основе его метода GetHashCode. Во всех остальных случаях хэш-код будет рассчитан как значение хэш-кода от имени типа данного значения.



## Файловые операции

Этот раздел содержит технические детали работы с классами файловых операций и описание соответствующих компонентов стандартной библиотеки MQL5.

Использование классов файловых операций позволит сэкономить время при разработке приложений использующих файловый ввод/вывод.

Стандартная библиотека MQL5 (в части файловых операций) размещается в рабочем каталоге терминала в папке `Include\Files`.

Имя класса	Назначение класса
<a href="#">CFile</a>	Базовый класс файловых операций
<a href="#">CFileBin</a>	Класс операций с бинарным файлом
<a href="#">CFileTxt</a>	Класс операций с текстовым файлом

## Класс CFile

Класс CFile является базовым классом для классов CFileBin и CFileTxt.

### Описание

Класс CFile обеспечивает своим потомкам доступ к общим функциям API MQL5 по работе с файлами и папками.

### Декларация

```
class CFile: public CObject
```

### Заголовок

```
#include <Files\File.mqh>
```

### Иерархия наследования

[CObject](#)

CFile

#### Прямые потомки

[CFileBin](#), [CFilePipe](#), [CFileTxt](#)

### Методы класса по группам

Атрибуты	
<a href="#">Handle</a>	Получает хэндл файла
<a href="#">Filename</a>	Получает имя файла
<a href="#">Flags</a>	Получает флаги файла
<a href="#">SetUnicode</a>	Устанавливает/сбрасывает флаг FILE_UNICODE
<a href="#">SetCommon</a>	Устанавливает/сбрасывает флаг FILE_COMMON
<b>Общие методы работы с файлами</b>	
<a href="#">Open</a>	Открывает файл
<a href="#">Close</a>	Закрывает файл
<a href="#">Delete</a>	Удаляет файл
<a href="#">IsExist</a>	Проверяет существование файла
<a href="#">Copy</a>	Копирует файл
<a href="#">Move</a>	Перемещает/переименовывает файл
<a href="#">Size</a>	Получает размер файла

<a href="#">Tell</a>	Получает текущее положение файлового указателя
<a href="#">Seek</a>	Устанавливает положение файлового указателя
<a href="#">Flush</a>	Сбрасывает на диск все данные
<a href="#">IsEnding</a>	Определяет конец файла
<a href="#">IsLineEnding</a>	Определяет конец строки
<b>Общие методы работы с папками</b>	
<a href="#">FolderCreate</a>	Создает папку
<a href="#">FolderDelete</a>	Удаляет папку
<a href="#">FolderClean</a>	Очищает папку
<b>Общие методы поиска файлов и папок</b>	
<a href="#">FileFindFirst</a>	Начинает поиск файлов
<a href="#">FileFindNext</a>	Продолжает поиск файлов
<a href="#">FileFindClose</a>	Закрывает хэндл поиска

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## Handle

Получает хэндл открытого файла.

```
int Handle()
```

### Возвращаемое значение

Хэндл привязанного к экземпляру класса открытого файла. Если нет привязанного файла возвращается -1.

## FileName

Получает имя открытого файла.

```
string FileName()
```

### Возвращаемое значение

Имя привязанного к экземпляру класса открытого файла. Если нет привязанного файла возвращается "".

## Flags

Получает флаги открытия файла.

```
int Flags()
```

### Возвращаемое значение

Флаги открытия привязанного к экземпляру класса файла.

## SetUnicode

Устанавливает/сбрасывает флаг FILE\_UNICODE.

```
void SetUnicode(  
    bool unicode           // значение флага  
)
```

### Параметры

*unicode*

[in] Новое значение флага FILE\_UNICODE.

### Примечание

От состояния флага FILE\_UNICODE зависит работа со строками. Если флаг сброшен, то используется кодировка ANSI (однобайтовые символы). Если флаг установлен, то используется кодировка UNICODE (двуихбайтовые символы). Если файл уже открыт, то флаг изменить нельзя.

## SetCommon

Устанавливает/сбрасывает флаг FILE\_COMMON.

```
void SetCommon(
    bool common           // значение флага
)
```

### Параметры

*common*

[in] Новое значение флага FILE\_COMMON.

### Примечание

От состояния флага FILE\_COMMON зависит, какая папка будет использоваться в качестве рабочей. Если флаг сброшен, то в качестве рабочей папки используется локальная папка терминала. Если флаг установлен, то в качестве рабочей папки используется общая папка. Если файл уже открыт, то флаг изменить нельзя.

## Open

Открывает указанный файл и, в случае удачного открытия, привязывает его к экземпляру класса.

```
int Open(
    const string file_name,           // имя файла
    int         flags,                // флаги
    short       delimiter=9          // разделитель
)
```

### Параметры

*file\_name*

[in] Имя открываемого файла.

*flags*

[in] Флаги открытия файла.

*delimiter=9*

[in] Разделитель для CSV-файла.

### Возвращаемое значение

Хэндл открытого файла.

### Примечание

Рабочая папка определяется ранее установленным/сброшенным при помощи метода SetCommon() флагом.

## Close

Закрывает файл, привязанный к экземпляру класса.

```
void Close()
```

## Delete

Удаляет файл, привязанный к экземпляру класса.

```
void Delete()
```

## Delete

Удаляет указанный файл.

```
void Delete(  
    const string file_name // имя файла  
)
```

### Параметры

*file\_name*  
[in] Имя файла для удаления.

### Примечание

Рабочая папка определяется ранее установленным/сброшенным при помощи метода SetCommon() флагом.

## IsExist

Проверяет существование файла.

```
bool IsExist(  
    const string file_name // имя файла  
)
```

### Параметры

*file\_name*  
[in] Имя проверяемого файла.

### Возвращаемое значение

true, если указанный файл существует.

## Copy

Копирует файл.

```
bool Copy(
    const string src_name,           // имя файла
    int       src_flag,             // флаг
    const string dst_name,           // имя файла
    int       dst_flags            // флаги
)
```

### Параметры

*src\_name*

[in] Имя файла-источника.

*src\_flag*

[in] Флаги файла-источника (используется только FILE\_COMMON).

*dst\_name*

[in] Имя файла-приемника.

*dst\_flags*

[in] Флаги файла-приемника (используются только FILE\_REWRITE и FILE\_COMMON).

### Возвращаемое значение

true - в случае удачи, false - если не удалось скопировать файл.

## Move

Перемещает/переименовывает файл.

```
bool Move(
    const string src_name,           // имя файла
    int       src_flag,             // флаг
    const string dst_name,           // имя файла
    int       dst_flags            // флаги
)
```

### Параметры

*src\_name*

[in] Имя файла-источника.

*src\_flag*

[in] Флаги файла-источника (используется только FILE\_COMMON).

*dst\_name*

[in] Имя файла-приемника.

*dst\_flags*

[in] Флаги файла-приемника (используются только FILE\_REWRITE и FILE\_COMMON).

### Возвращаемое значение

true - в случае удачи, false - если не удалось переместить/переименовать файл.

## Size

Получает размер файла в байтах.

```
ulong Size()
```

### Возвращаемое значение

размер файла в байтах. Если нет привязанного файла возвращается ULONG\_MAX.

## Tell

Получает текущее положение файлового указателя.

```
ulong Tell()
```

### Возвращаемое значение

текущее положение файлового указателя. Если нет привязанного файла возвращается `ULONG_MAX`.

## Seek

Устанавливает положение файлового указателя.

```
void Seek(  
    long           offset,      // смещение  
    ENUM_FILE_POSITION origin   // точка отсчета  
)
```

### Параметры

*offset*

[in] Смещение в байтах (может принимать и отрицательное значение).

*origin*

[in] Точка отсчета для смещения.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить файловый указатель.

## Flush

Сбрасывает на диск все данных, оставшиеся в файловом буфере ввода-вывода.

```
void Flush()
```

## IsEnding

Определяет конец файла в процессе чтения.

```
bool IsEnding()
```

### Возвращаемое значение

true в случае, если в процессе чтения или перемещения файлового указателя достигнут конец файла.

## IsLineEnding

Определяет конец строки в текстовом файле в процессе чтения.

```
bool IsLineEnding()
```

### Возвращаемое значение

true в случае, если в процессе чтения txt или csv-файла достигнут конец строки (символы CR-LF).

## FolderCreate

Создает новую папку.

```
bool FolderCreate(
    const string folder_name           // имя папки
)
```

### Параметры

*folder\_name*

[in] Имя папки, которую требуется создать. Содержит путь к папке (относительно рабочей папки, заданной при помощи флага FILE\_COMMON).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать папку.

### Примечание

Рабочая папка определяется ранее установленным/сброшенным при помощи метода SetCommon() флагом.

## FolderDelete

Удаляет указанную папку.

```
bool FolderDelete(
    const string folder_name           // имя папки
)
```

### Параметры

*folder\_name*

[in] Имя папки, которую требуется удалить. Содержит путь к папке (относительно рабочей папки, заданной при помощи флага FILE\_COMMON).

### Возвращаемое значение

true - в случае удачи, false - если не удалось удалить папку.

### Примечание

Рабочая папка определяется ранее установленным/сброшенным при помощи метода SetCommon() флагом.

## FolderClean

Очищает указанную папку.

```
bool FolderClean(
    const string folder_name           // имя папки
)
```

### Параметры

*folder\_name*

[in] Имя папки, которую требуется очистить. Содержит путь к папке (относительно рабочей папки, заданной при помощи флага FILE\_COMMON).

### Возвращаемое значение

true - в случае удачи, false - если не удалось очистить папку.

### Примечание

Рабочая папка определяется ранее установленным/сброшенным при помощи метода SetCommon() флагом.

## FileFindFirst

Начинает поиск файлов в соответствии с указанным фильтром.

```
int FileFindFirst(
    const string filter,           // фильтр поиска
    string& file_name            // ссылка
)
```

### Параметры

*filter*

[in] Фильтр поиска.

*file\_name*

[out] Ссылка на строку, куда в случае удачи помещается имя первого найденного файла.

### Возвращаемое значение

Хэндл, который необходимо использовать для дальнейшего поиска файлов методом FileFindNext(), либо INVALID\_HANDLE в случае, когда нет ни одного файла, соответствующего фильтру.

### Примечание

Рабочая папка определяется ранее установленным/сброшенным при помощи метода SetCommon() флагом.

## FileFindNext

Продолжает поиск, начатый функцией FileFindFirst().

```
bool FileFindNext(
    int      search_handle,      // хэндл поиска
    string&  file_name         // ссылка
)
```

### Параметры

*search\_handle*

[in] Хэндл поиска, полученный методом FileFindFirst().

*file\_name*

[in] Ссылка на строку, куда в случае удачи помещается имя найденного файла.

### Возвращаемое значение

true - в случае удачи, false - если больше нет ни одного файла, соответствующего фильтру.

## FileFindClose

Закрывает хэндл поиска.

```
void FileFindClose(
    int search_handle // хэндл поиска
)
```

### Параметры

*search\_handle*

[in] Хэндл поиска, полученный методом FileFindFirst().

## Класс CFileBin

Класс CFileBin является классом для упрощенного доступа к двоичным файлам.

### Описание

Класс CFileBin обеспечивает доступ к двоичным файлам.

### Декларация

```
class CFileBin: public CFile
```

### Заголовок

```
#include <Files\FileBin.mqh>
```

### Иерархия наследования

```
CObject  
CFile  
CFileBin
```

### Методы класса по группам

Открытие	
<a href="#">Open</a>	Открывает двоичный файл
Методы записи	
<a href="#">WriteChar</a>	Записывает переменную типа char или uchar
<a href="#">WriteShort</a>	Записывает переменную типа short или ushort
<a href="#">WriteInteger</a>	Записывает переменную типа int или uint
<a href="#">WriteLong</a>	Записывает переменную типа long или ulong
<a href="#">WriteFloat</a>	Записывает переменную типа float
<a href="#">WriteDouble</a>	Записывает переменную типа double
<a href="#">WriteString</a>	Записывает переменную типа string
<a href="#">WriteCharArray</a>	Записывает массив переменных типа char или uchar
<a href="#">WriteShortArray</a>	Записывает массив переменных типа short или ushort
<a href="#">WriteIntegerArray</a>	Записывает массив переменных типа int или uint
<a href="#">WriteLongArray</a>	Записывает массив переменных типа long или ulong

<a href="#">WriteFloatArray</a>	Записывает массив переменных типа float
<a href="#">WriteDoubleArray</a>	Записывает массив переменных типа double
<a href="#">WriteObject</a>	Записывает данные экземпляра наследника класса CObject
<b>Методы чтения</b>	
<a href="#">ReadChar</a>	Читает переменную типа char или uchar
<a href="#">ReadShort</a>	Читает переменную типа short или ushort
<a href="#">ReadInteger</a>	Читает переменную типа int или uint
<a href="#">ReadLong</a>	Читает переменную типа long или ulong
<a href="#">ReadFloat</a>	Читает переменную типа float
<a href="#">ReadDouble</a>	Читает переменную типа double
<a href="#">ReadString</a>	Читает переменную типа string
<a href="#">ReadCharArray</a>	Читает массив переменных типа char или uchar
<a href="#">ReadShortArray</a>	Читает массив переменных типа short или ushort
<a href="#">ReadIntegerArray</a>	Читает массив переменных типа int или uint
<a href="#">ReadLongArray</a>	Читает массив переменных типа long или ulong
<a href="#">ReadFloatArray</a>	Читает массив переменных типа float
<a href="#">ReadDoubleArray</a>	Читает массив переменных типа double
<a href="#">ReadObject</a>	Читает данные экземпляра наследника класса CObject

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

**Методы унаследованные от CFile**

[Handle](#), [FileName](#), [Flags](#), [SetUnicode](#), [SetCommon](#), [Open](#), [Close](#), [Delete](#), [Size](#), [Tell](#), [Seek](#), [Flush](#), [IsEnding](#), [IsLineEnding](#), [Delete](#), [IsExist](#), [Copy](#), [Move](#), [FolderCreate](#), [FolderDelete](#), [FolderClean](#), [FileFindFirst](#), [FileFindNext](#), [FileFindClose](#)

## Open

Открывает указанный двоичный файл и, в случае удачного открытия, привязывает его к экземпляру класса.

```
int Open(
    const string file_name,      // имя файла
    int           flags         // флаги
)
```

### Параметры

*file\_name*

[in] Имя открываемого файла.

*flags*

[in] Флаги открытия файла (флаг FILE\_BIN устанавливается принудительно).

### Возвращаемое значение

Хэндл открытого файла.

## WriteChar

Записывает в файл переменную типа char или uchar.

```
uint WriteChar(  
    char value        // значение  
)
```

### Параметры

*value*

[in] Переменная для записи.

### Возвращаемое значение

Количество записанных байтов.

## WriteShort

Записывает в файл переменную типа short или ushort.

```
uint WriteShort(
    short value           // значение
)
```

### Параметры

*value*

[in] Переменная для записи.

### Возвращаемое значение

Количество записанных байтов.

## WriteInteger

Записывает в файл переменную типа int или uint.

```
uint WriteInteger(  
    int value // значение  
)
```

### Параметры

*value*

[in] Переменная для записи.

### Возвращаемое значение

Количество записанных байтов.

## WriteLong

Записывает в файл переменную типа long или ulong.

```
uint WriteLong(  
    long value // значение  
)
```

### Параметры

*value*

[in] Переменная для записи.

### Возвращаемое значение

Количество записанных байтов.

## WriteFloat

Записывает в файл переменную типа float.

```
uint WriteFloat(
    float value          // значение
)
```

### Параметры

*value*

[in] Переменная для записи.

### Возвращаемое значение

Количество записанных байтов.

## WriteDouble

Записывает в файл переменную типа double.

```
uint WriteDouble(  
    double value           // значение  
)
```

### Параметры

*value*

[in] Переменная для записи.

### Возвращаемое значение

Количество записанных байтов.

## WriteString

Записывает в файл переменную типа string.

```
uint WriteString(
    const string value           // значение
)
```

### Параметры

*value*

[in] Стока для записи.

### Возвращаемое значение

Количество записанных байтов.

## WriteString

Записывает в файл переменную типа string.

```
uint WriteString(
    const string value,          // значение
    int         size             // размер
)
```

### Параметры

*value*

[in] Стока для записи.

*size*

[in] Количество символов для записи.

### Возвращаемое значение

Количество записанных байтов.

## WriteCharArray

Записывает в файл массив переменных типа char или uchar.

```
uint WriteCharArray(
    char& array[],           // массив
    int     start_item=0,      // стартовый элемент
    int     items_count=-1    // количество элементов
)
```

### Параметры

*array[]*

[in] Массив для записи.

*start\_item=0*

[in] Стартовый элемент для записи.

*items\_count=-1*

[in] Количество элементов для записи (-1 - весь массив).

### Возвращаемое значение

Количество записанных байтов.

## WriteShortArray

Записывает в файл массив переменных типа short или ushort.

```
uint WriteShortArray(
    short& array[],           // массив
    int     start_item=0,      // стартовый элемент
    int     items_count=-1    // количество элементов
)
```

### Параметры

*array[]*

[in] Массив для записи.

*start\_item=0*

[in] Стартовый элемент для записи.

*items\_count=-1*

[in] Количество элементов для записи (-1 - весь массив).

### Возвращаемое значение

Количество записанных байтов.

## WriteIntegerArray

Записывает в файл массив переменных типа int или uint.

```
uint WriteIntegerArray(
    int& array[],           // массив
    int start_item=0,        // стартовый элемент
    int items_count=-1      // количество элементов
)
```

### Параметры

*array[]*

[in] Массив для записи.

*start\_item=0*

[in] Стартовый элемент для записи.

*items\_count=-1*

[in] Количество элементов для записи (-1 - весь массив).

### Возвращаемое значение

Количество записанных байтов.

## WriteLongArray

Записывает в файл массив переменных типа long или ulong.

```
uint WriteLongArray(
    long& array[],           // массив
    int     start_item=0,      // стартовый элемент
    int     items_count=-1    // количество элементов
)
```

### Параметры

*array[]*

[in] Массив для записи.

*start\_item=0*

[in] Стартовый элемент для записи.

*items\_count=-1*

[in] Количество элементов для записи (-1 - весь массив).

### Возвращаемое значение

Количество записанных байтов.

## WriteFloatArray

Записывает в файл массив переменных типа float.

```
uint WriteFloatArray(
    float& array[],           // массив
    int     start_item=0,      // стартовый элемент
    int     items_count=-1    // количество элементов
)
```

### Параметры

*array[]*

[in] Массив для записи.

*start\_item=0*

[in] Стартовый элемент для записи.

*items\_count=-1*

[in] Количество элементов для записи (-1 - весь массив).

### Возвращаемое значение

Количество записанных байтов.

## WriteDoubleArray

Записывает в файл массив переменных типа double.

```
uint WriteDoubleArray(
    double& array[],           // массив
    int      start_item=0,      // стартовый элемент
    int      items_count=-1    // количество элементов
)
```

### Параметры

*array[]*

[in] Массив для записи.

*start\_item=0*

[in] Стартовый элемент для записи.

*items\_count=-1*

[in] Количество элементов для записи (-1 - весь массив).

### Возвращаемое значение

Количество записанных байтов.

## WriteObject

Записывает в файл данные экземпляра наследника класса CObject.

```
bool WriteObject(
    CObject* object      // указатель
)
```

### Параметры

*object*

[in] Указатель на экземпляр наследника класса CObject для записи.

### Возвращаемое значение

true - в случае удачи, false - если не удалось записать данные.

## ReadChar

Читает из файла переменную типа char или uchar.

```
bool ReadChar(  
    char& value // значение флага  
)
```

### Параметры

*value*

[in] Ссылка на переменную для размещения прочитанных данных.

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadShort

Читает из файла переменную типа short или ushort.

```
bool ReadShort(  
    short& value  
)
```

### Параметры

*value*

[in] Ссылка на переменную для размещения прочитанных данных.

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadInteger

Читает из файла переменную типа int или uint.

```
bool ReadInteger(  
    int& value // переменная  
)
```

### Параметры

*value*

[in] Ссылка на переменную для размещения прочитанных данных.

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadLong

Читает из файла переменную типа long или ulong.

```
bool ReadLong(  
    long& value  
)
```

### Параметры

*value*

[in] Ссылка на переменную для размещения прочитанных данных.

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadFloat

Читает из файла переменную типа float.

```
bool ReadFloat(  
    float& value // переменная  
)
```

### Параметры

*value*

[in] Ссылка на переменную для размещения прочитанных данных.

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadDouble

Читает из файла переменную типа double.

```
bool ReadDouble(  
    double& value  
)
```

### Параметры

*value*

[in] Ссылка на переменную для размещения прочитанных данных.

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadString

Читает из файла переменную типа string.

```
bool ReadString(  
    string& value          // строка  
)
```

### Параметры

*value*

[in] Ссылка на строку для размещения прочитанных данных.

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadString

Читает из файла переменную типа string.

```
bool ReadString(  
    string& value  
)
```

### Параметры

*value*

[in] Ссылка на строку для размещения прочитанных данных.

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadCharArray

Читает из файла массив переменных типа char или uchar.

```
bool ReadCharArray(
    char& array[],           // массив
    int     start_item=0,      // стартовый элемент
    int     items_count=-1    // количество элементов
)
```

### Параметры

*array[]*

[in] Ссылка на массив для размещения прочитанных данных.

*start\_item=0*

[in] Стартовый элемент для чтения.

*items\_count=-1*

[in] Количество элементов для чтения (-1 - до конца файла).

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось прочитать данные.

## ReadShortArray

Читает из файла массив переменных типа short или ushort.

```
bool ReadShortArray(
    short& array[],           // массив
    int     start_item=0,      // стартовый элемент
    int     items_count=-1    // количество элементов
)
```

### Параметры

*array[]*

[in] Ссылка на массив для размещения прочитанных данных.

*start\_item=0*

[in] Стартовый элемент для чтения.

*items\_count=-1*

[in] Количество элементов для чтения (-1 - конца файла).

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadIntegerArray

Читает из файла массив переменных типа int или uint.

```
bool ReadIntegerArray(
    int& array[],           // массив
    int start_item=0,        // стартовый элемент
    int items_count=-1       // количество элементов
)
```

### Параметры

*array[]*

[in] Ссылка на массив для размещения прочитанных данных.

*start\_item=0*

[in] Стартовый элемент для чтения.

*items\_count=-1*

[in] Количество элементов для чтения (-1 - конца файла).

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadLongArray

Читает из файла массив переменных типа long или ulong.

```
bool ReadLongArray(
    long& array[],           // массив
    int    start_item=0,      // стартовый элемент
    int    items_count=-1    // количество элементов
)
```

### Параметры

*array[]*

[in] Ссылка на массив для размещения прочитанных данных.

*start\_item=0*

[in] Стартовый элемент для чтения.

*items\_count=-1*

[in] Количество элементов для чтения (-1 - конца файла).

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadFloatArray

Читает из файла массив переменных типа float.

```
bool ReadFloatArray(
    float& array[],           // массив
    int     start_item=0,      // стартовый элемент
    int     items_count=-1    // количество элементов
)
```

### Параметры

*array[]*

[in] Ссылка на массив для размещения прочитанных данных.

*start\_item=0*

[in] Стартовый элемент для чтения.

*items\_count=-1*

[in] Количество элементов для чтения (-1 - конца файла).

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## ReadDoubleArray

Читает из файла массив переменных типа double.

```
bool ReadDoubleArray(
    double& array[],           // массив
    int      start_item=0,      // стартовый элемент
    int      items_count=-1    // количество элементов
)
```

### Параметры

*array[]*

[in] Ссылка на массив для размещения прочитанных данных.

*start\_item=0*

[in] Стартовый элемент для чтения.

*items\_count=-1*

[in] Количество элементов для чтения (-1 - конца файла).

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось прочитать данные.

## ReadObject

Читает из файла данные экземпляра наследника класса CObject.

```
bool ReadObject(
    CObject* object      // указатель
)
```

### Параметры

*object*

[in] Указатель на экземпляр наследника класса CObject для чтения.

### Возвращаемое значение

true - в случае удачи, false - если не удалось прочитать данные.

## Класс CFileTxt

Класс CFileTxt является классом для упрощенного доступа к текстовым файлам.

### Описание

Класс CFileTxt обеспечивает доступ к текстовым файлам.

### Декларация

```
class CFileTxt: public CFile
```

### Заголовок

```
#include <Files\FileTxt.mqh>
```

### Иерархия наследования

```
CObject  
CFile  
CFileTxt
```

### Методы класса по группам

Открытие	
<a href="#">Open</a>	Открывает текстовый файл
Методы записи	
<a href="#">WriteString</a>	Записывает переменную типа string
Методы чтения	
<a href="#">ReadString</a>	Читает переменную типа string

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CFile

[Handle](#), [FileName](#), [Flags](#), [SetUnicode](#), [SetCommon](#), [Open](#), [Close](#), [Delete](#), [Size](#), [Tell](#), [Seek](#), [Flush](#), [IsEnding](#), [IsLineEnding](#), [Delete](#), [IsExist](#), [Copy](#), [Move](#), [FolderCreate](#), [FolderDelete](#), [FolderClean](#), [FileFindFirst](#), [FileFindNext](#), [FileFindClose](#)

## Open

Открывает указанный текстовый файл и, в случае удачного открытия, привязывает его к экземпляру класса.

```
int Open(
    const string file_name,      // имя файла
    int           flags         // флаги
)
```

### Параметры

*file\_name*

[in] Имя открываемого файла.

*flags*

[in] Флаги открытия файла (флаг FILE\_TXT устанавливается принудительно).

### Возвращаемое значение

Хэндл открытого файла.

## WriteString

Записывает в файл переменную типа string.

```
uint WriteString(
    const string value      // строка
)
```

### Параметры

*value*

[in] Стока для записи.

### Возвращаемое значение

Количество записанных байтов.

## ReadString

Читает из файла переменную типа string.

```
string ReadString()
```

### Возвращаемое значение

Считанная строка.

## Операции со строками

Этот раздел содержит технические детали работы с классом строковых операций и описание соответствующих компонентов стандартной библиотеки MQL5.

Использование класса строковых операций позволит сэкономить время при создании пользовательских приложений обрабатывающих текстовую информацию.

Стандартная библиотека MQL5 (в части работы с строками) размещается в рабочем каталоге терминала в папке `Include\Strings`.

Имя класса	Назначение класса
<a href="#">CString</a>	Класс строковых операций

## Класс CString

Класс CString является классом для упрощенного доступа к переменным типа string.

### Описание

Класс CString обеспечивает упрощенный доступ к функциям API MQL5 по работе с переменными типа string.

### Декларация

```
class CString: public CObject
```

### Заголовок

```
#include <Strings\String.mqh>
```

### Иерархия наследования

[CObject](#)

CString

### Методы класса по группам

Методы доступа к данным	
<a href="#">Str</a>	Получает строку
<a href="#">Len</a>	Получает длину строки
<a href="#">Copy</a>	Получает копию строки
Методы наполнения	
<a href="#">Fill</a>	Заполняет строку
<a href="#">Assign</a>	Присваивает строку
<a href="#">Append</a>	Добавляет строку
<a href="#">Insert</a>	Вставляет строку
Методы сравнения	
<a href="#">Compare</a>	Сравнивает строки
<a href="#">CompareNoCase</a>	Сравнивает строки без учета регистра
Методы работы с подстроками	
<a href="#">Left</a>	Получает подстроку слева
<a href="#">Right</a>	Получает подстроку справа
<a href="#">Mid</a>	Получает подстроку из середины
Методы обрезания/удаления	

<a href="#">Trim</a>	Обрезает строку слева и справа
<a href="#">TrimLeft</a>	Обрезает строку слева
<a href="#">TrimRight</a>	Обрезает строку справа
<a href="#">Clear</a>	Очищает строку
<b>Методы преобразования</b>	
<a href="#">ToUpper</a>	Переводит строку в верхний регистр
<a href="#">ToLower</a>	Переводит строку в нижний регистр
<a href="#">Reverse</a>	Разворачивает строку
<b>Методы поиска</b>	
<a href="#">Find</a>	Ищет подстроку слева-направо
<a href="#">FindRev</a>	Ищет подстроку справа-налево
<a href="#">Remove</a>	Удаляет подстроку
<a href="#">Replace</a>	Заменяет подстроку

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#)

## Str

Получает строку.

```
string Str() const;
```

### Возвращаемое значение

Копия строки.

## Len

Получает длину строки.

```
uint Len() const;
```

### Возвращаемое значение

Длина строки.

## Copy

Копирует строку по ссылке.

```
void Copy(  
    string& copy      // ссылка  
) const;
```

### Параметры

*copy*

[in] Ссылка на строку для данных.

## Copy

Копирует строку в экземпляр класса CString.

```
void Copy(  
    CString* copy      // указатель  
) const;
```

### Параметры

*copy*

[in] Указатель на экземпляр класса CString для данных.

## Fill

Заполняет строку указанным символом.

```
bool Fill(
    short character      // символ
)
```

### Параметры

*character*

[in] Символ для заполнения строки.

### Возвращаемое значение

true - в случае удачи, false - если не удалось заполнить строку.

## Assign

Присваивает строку.

```
void Assign(
    const string str      // строка
)
```

### Параметры

*str*

[in] Стока для присвоения.

## Assign

Присваивает строку из экземпляра класса `CString`.

```
void Assign(
    CString* str      // указатель
)
```

### Параметры

*str*

[in] Указатель на экземпляр класса `CString` для присвоения.

## Append

Добавляет строку.

```
void Append(
    const string str      // строка
)
```

### Параметры

*str*

[in] Стока для добавления.

## Append

Добавляет строку из экземпляра класса `CString`.

```
void Append(
    CString* string      // указатель
)
```

### Параметры

*string*

[in] Указатель на экземпляр класса `CString` для добавления.

## Insert

Вставляет строку в указанную позицию.

```
uint Insert(
    uint      pos,      // позиция
    const string str     // строка
)
```

### Параметры

*pos*

[in] Позиция для вставки.

*str*

[in] Стока для вставки.

### Возвращаемое значение

Длина результирующей строки.

## Insert

Вставляет строку в указанную позицию из экземпляра класса `CString`.

```
uint Insert(
    uint      pos,      // позиция
    CString* str        // указатель
)
```

### Параметры

*pos*

[in] Позиция для вставки.

*str*

[in] Указатель на экземпляр класса `CString` для вставки.

### Возвращаемое значение

Длина результирующей строки.

## Compare

Сравнивает со строкой.

```
int Compare(
    const string str      // строка
) const;
```

### Параметры

*str*

[in] Стока для сравнения.

### Возвращаемое значение

0 - если строки равны, -1 - если строка-член класса меньше чем строка-параметр, 1 - если строка-член класса больше чем строка-параметр.

## Compare

Сравнивает со строкой экземпляра класса *CString*.

```
int Compare(
    CString* str      // указатель
) const;
```

### Параметры

*str*

[in] Указатель на экземпляр класса *CString* для сравнения.

### Возвращаемое значение

0 - если строки равны, -1 - если строка-член класса меньше чем строка-параметр, 1 - если строка-член класса больше чем строка-параметр.

## CompareNoCase

Сравнивает со строкой без учета регистра.

```
int CompareNoCase(
    const string str      // строка
) const;
```

### Параметры

*str*

[in] Стока для сравнения.

### Возвращаемое значение

0 - если строки равны, -1 - если строка-член класса меньше чем строка-параметр, 1 - если строка-член класса больше чем строка-параметр.

## CompareNoCase

Сравнивает со строкой экземпляра класса *CString* без учета регистра.

```
int CompareNoCase(
    CString* str      // указатель
) const;
```

### Параметры

*str*

[in] Указатель на экземпляр класса *CString* для сравнения.

### Возвращаемое значение

0 - если строки равны, -1 - если строка-член класса меньше чем строка-параметр, 1 - если строка-член класса больше чем строка-параметр.

## Left

Получает подстроку указанной длины с начала строки.

```
string Left(
    uint count           // длина
)
```

### Параметры

*count*

[in] Длина подстроки.

### Возвращаемое значение

Результирующая подстрока.

## Right

Получает подстроку указанной длины с конца строки.

```
string Right(
    uint count           // длина
)
```

### Параметры

*count*

[in] Длина подстроки.

### Возвращаемое значение

Результирующая подстрока.

## Mid

Получает подстроку указанной длины с указанной позиции строки.

```
string Mid(
    uint pos,           // позиция
    uint count          // длина
)
```

### Параметры

*pos*

[in] Позиция подстроки.

*count*

[in] Длина подстроки.

### Возвращаемое значение

Результирующая подстрока.

## Trim

Удаляет из строки все символы входящие в набор (дополнительно ' ', '\t', '\r', '\n') в начале и в конце строки.

```
int Trim(  
    const string targets // набор  
)
```

### Параметры

*targets*  
[in] Набор символов для удаления.

### Возвращаемое значение

Количество удаленных символов.

### Пример:

```
//--- example for CString::Trim  
#include <Strings\String.mqh>  
//---  
void OnStart()  
{  
    CString str;  
    //---  
    str.Assign(" \t\tABCD\r\n");  
    printf("Source string '%s'",str.Str());  
    //---  
    str.Trim("DA-DA-DA");  
    printf("Result string '%s'",str.Str());  
}
```

## TrimLeft

Удаляет из строки все символы входящие в набор (дополнительно ''','\t','\r','\n') в начале строки.

```
int TrimLeft(
    const string targets           // набор
)
```

### Параметры

*targets*

[in] Набор символов для удаления.

### Возвращаемое значение

Количество удаленных символов.

## TrimRight

Удаляет из строки все символы входящие в набор (дополнительно ''','\t','\r','\n') в конце строки.

```
int TrimRight(
    const string targets           // набор
)
```

### Параметры

*targets*

[in] Набор символов для удаления.

### Возвращаемое значение

Количество удаленных символов.

## Clear

Очищает строку

```
bool Clear()
```

### Возвращаемое значение

true - в случае удачи, false - если не удалось очистить строку.

## ToUpper

Переводит все символы строки в верхний регистр.

```
bool ToUpper()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить регистр.

## ToLower

Переводит все символы строки в нижний регистр.

```
bool ToLower()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить регистр.

## Reverse

Разворачивает строку (меняет местами попарно начальные и конечные символы).

```
void Reverse()
```

## Find

Ищет первое включение подстроки с указанной позиции.

```
int Find(
    uint      start,           // позиция
    const string substring     // подстрока
) const;
```

### Параметры

*start*

[in] Стартовая позиция поиска подстроки.

*substring*

[in] Образец подстроки для поиска.

### Возвращаемое значение

Индекс первого включения подстроки (-1 если подстрока не найдена).

## FindRev

Ищет последнее включение подстроки.

```
int FindRev(  
    const string substring // подстрока  
) const;
```

### Параметры

*substring*

[in] Образец подстроки для поиска.

### Возвращаемое значение

Индекс последнего включения подстроки (-1 если подстрока не найдена).

## Remove

Удаляет все включения подстроки.

```
uint Remove(
    const string substring      // подстрока
)
```

### Параметры

*substring*

[in] Образец подстроки для поиска.

### Возвращаемое значение

Количество удалений подстроки.

## Replace

Заменяет все включения подстроки.

```
uint Replace(
    const string substring,      // подстрока
    const string newstring       // подстрока
)
```

### Параметры

*substring*

[in] Образец подстроки для поиска.

*newstring*

[in] Образец подстроки для замены.

### Возвращаемое значение

Количество замен подстроки.

## Графические объекты

Этот раздел содержит технические детали работы с классами графических объектов и описание соответствующих компонентов стандартной библиотеки MQL5.

Использование классов графических объектов позволит сэкономить время при создании пользовательских программ (скриптов, экспертов).

Стандартная библиотека MQL5 (в части графических объектов) размещается в рабочем каталоге терминала в папке `Include\ChartObjects`.

Класс/группа	Описание
<a href="#"><u>Базовый класс для работы с графическим объектом CChartObject</u></a>	Базовый класс графического объекта
<a href="#"><u>Линии</u></a>	Группа классов "Линии"
<a href="#"><u>Каналы</u></a>	Группа классов "Каналы"
<a href="#"><u>Инструменты Ганна</u></a>	Группа классов "Ганн"
<a href="#"><u>Инструменты Фибоначчи</u></a>	Группа классов "Фибоначчи"
<a href="#"><u>Инструменты Эллиотта</u></a>	Группа классов "Эллиотт"
<a href="#"><u>Фигуры</u></a>	Группа классов "Фигуры"
<a href="#"><u>Стрелки</u></a>	Группа классов "Стрелки"
<a href="#"><u>Элементы управления</u></a>	Группа классов "Управление"

## CChartObject

Класс CChartObject является базовым классом для классов графических объектов чарта стандартной библиотеки MQL5.

### Описание

Класс CChartObject обеспечивает всем своим потомкам возможность упрощенного доступа к функциям API MQL5.

### Декларация

```
class CChartObject : public CObject
```

### Заголовок

```
#include <ChartObjects\ChartObject.mqh>
```

### Иерархия наследования

[CObject](#)

CChartObject

#### Прямые потомки

[CChartObjectArrow](#), [CChartObjectBitmap](#), [CChartObjectBmpLabel](#), [CChartObjectCycles](#),  
[CChartObjectElliottWave3](#), [CChartObjectEllipse](#), [CChartObjectFiboArc](#), [CChartObjectFiboFan](#),  
[CChartObjectFiboTimes](#), [CChartObjectHLine](#), [CChartObjectRectangle](#), [CChartObjectSubChart](#),  
[CChartObjectText](#), [CChartObjectTrend](#), [CChartObjectTriangle](#), [CChartObjectVLine](#)

### Методы класса по группам

Атрибуты	
<a href="#">ChartId</a>	Получает идентификатор чарта, которому принадлежит графический объект
<a href="#">Window</a>	Получает номер окна чарта в котором находится графический объект
<a href="#">Name</a>	Получает/устанавливает наименование графического объекта
<a href="#">NumPoints</a>	Получает количество точек привязки
<b>Наполнение</b>	
<a href="#">Attach</a>	Привязывает графический объект чарта
<a href="#">SetPoint</a>	Устанавливает параметры точки привязки
<b>Удаление</b>	
<a href="#">Delete</a>	Удаляет графический объект чарта

<a href="#"><u>Detach</u></a>	Отвязывает графический объект чарта
<b>Перемещение</b>	
<a href="#"><u>ShiftObject</u></a>	Относительное перемещение объекта
<a href="#"><u>ShiftPoint</u></a>	Относительное перемещение точки объекта
<b>Свойства объекта</b>	
<a href="#"><u>Time</u></a>	Получает/устанавливает временную координату точки объекта
<a href="#"><u>Price</u></a>	Получает/устанавливает ценовую координату точки объекта
<a href="#"><u>Color</u></a>	Получает/устанавливает цвет объекта
<a href="#"><u>Style</u></a>	Получает/устанавливает стиль линии объекта
<a href="#"><u>Width</u></a>	Получает/устанавливает ширину линии объекта
<a href="#"><u>Background</u></a>	Получает/устанавливает флаг рисования объекта фоном
<a href="#"><u>Selected</u></a>	Получает/устанавливает флаг выбранности объекта
<a href="#"><u>Selectable</u></a>	Получает/устанавливает флаг выбираемости объекта
<a href="#"><u>Description</u></a>	Получает/устанавливает текст объекта
<a href="#"><u>Tooltip</u></a>	Получает/устанавливает текст всплывающей подказки объекта
<a href="#"><u>Timeframes</u></a>	Получает/устанавливает маску флагов видимости объекта
<a href="#"><u>Z_Order</u></a>	Получает/устанавливает приоритет графического объекта на получение события нажатия мышки на графике
<a href="#"><u>CreateTime</u></a>	Получает время создания объекта
<b>Свойства уровней объекта</b>	
<a href="#"><u>LevelsCount</u></a>	Получает/устанавливает количество уровней объекта
<a href="#"><u>LevelColor</u></a>	Получает/устанавливает цвет линии уровня
<a href="#"><u>LevelStyle</u></a>	Получает/устанавливает стиль линии уровня
<a href="#"><u>LevelWidth</u></a>	Получает/устанавливает ширину линии уровня
<a href="#"><u>LevelValue</u></a>	Получает/устанавливает значение уровня

<a href="#"><u>LevelDescription</u></a>	Получает/устанавливает текст уровня
<b>Доступ к функциям API MQL5</b>	
<a href="#"><u>GetInteger</u></a>	Получает значение свойства объекта
<a href="#"><u>SetInteger</u></a>	Устанавливает значение свойства объекта
<a href="#"><u>GetDouble</u></a>	Получает значение свойства объекта
<a href="#"><u>SetDouble</u></a>	Устанавливает значение свойства объекта
<a href="#"><u>GetString</u></a>	Получает значение свойства объекта
<a href="#"><u>SetString</u></a>	Устанавливает значение свойства объекта
<b>Ввод/вывод</b>	
<b>virtual <a href="#"><u>Save</u></a></b>	Виртуальный метод записи в файл
<b>virtual <a href="#"><u>Load</u></a></b>	Виртуальный метод чтения из файла
<b>virtual <a href="#"><u>Type</u></a></b>	Виртуальный метод идентификации

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

## ChartId

Получает идентификатор чарта, которому принадлежит графический объект.

```
long ChartId() const
```

### Возвращаемое значение

Идентификатор чарта, на котором находится графический объект. Если нет привязанного объекта, возвращается -1.

### Пример:

```
//--- example for CChartObject::ChartId
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get chart idintifier of chart object
    long chart_id=object.ChartId();
}
```

## Window

Получает номер окна чарта в котором находится графический объект.

```
int Window() const
```

### Возвращаемое значение

Номер окна чарта в котором находится графический объект (0 - основное окно). Если нет привязанного объекта, возвращается -1.

### Пример:

```
//--- example for CChartObject::Window
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get window of chart object
    int window=object.Window();
}
```

## Name (метод Get)

Получает наименование графического объекта.

```
string Name() const
```

### Возвращаемое значение

Наименование графического объекта привязанного к экземпляру класса. Если нет привязанного объекта возвращается NULL.

## Name (метод Set)

Устанавливает наименование графического объекта.

```
bool Name(
    string name // новое наименование
)
```

### Параметры

*name*  
 [in] Новое наименование графического объекта.

### Возвращаемое значение

true в случае успеха, false - если не удалось изменить наименование.

### Пример:

```
//--- example for CChartObject::Name
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get name of chart object
    string object_name=object.Name();
    if(object_name!="MyChartObject")
    {
        //--- set name of chart object
        object.Name("MyChartObject");
    }
}
```

## NumPoints

Получает количество точек привязки графического объекта.

```
int NumPoints() const
```

### Возвращаемое значение

Количество точек привязки графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

### Пример:

```
//--- example for CChartObject::NumPoints
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get points count of chart object
    int points=object.NumPoints();
}
```

## Attach

Привязывает графический объект к экземпляру класса.

```
bool Attach(
    long    chart_id,      // идентификатор чарта
    string  name,         // имя объекта
    int     window,        // окно чарта
    int     points         // количество точек
)
```

### Параметры

*chart\_id*

[out] Идентификатор чарта.

*name*

[in] Наименование (имя) графического объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*points*

[in] Количество точек привязки графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если нет возможности привязать объект.

### Пример:

```
//--- example for CChartObject::Attach
#include <ChartObjects\ChartObject.mqh>
//---

void OnStart()
{
    CChartObject object;
    //--- attach chart object
    if(!object.Attach(ChartID(),"MyObject",0,2))
    {
        printf("Object attach error");
        return;
    }
}
```

## SetPoint

Устанавливает новые координаты указанной точки привязки графического объекта.

```
bool SetPoint(
    int      point,          // номер точки
    datetime new_time,       // координата времени
    double   new_price       // координата цены
)
```

### Параметры

*point*

[in] Номер точки привязки графического объекта.

*new\_time*

[in] Новое значение координаты времени указанной точки привязки

*new\_price*

[in] Новое значение координаты цены указанной точки привязки.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить координаты точки.

### Пример:

```
//--- example for CChartObject::SetPoint
#include <ChartObjects\ChartObject.mqh>
//---

void OnStart()
{
    CChartObject object;
    double price;
//---
    if(object.NumPoints()>0)
    {
        //--- set point of chart object
        object.SetPoint(0,CurrTime(),price);
    }
}
```

## Delete

Удаляет привязанный графический объект с чарта.

```
bool Delete()
```

### Возвращаемое значение

true - в случае удачи, false - если нет возможности удалить объект.

### Пример:

```
//--- example for CChartObject::Delete
#include <ChartObjects\ChartObject.mqh>
//---

void OnStart()
{
    CChartObject object;
    //--- detach chart object
    if(!object.Delete())
    {
        printf("Object delete error");
        return;
    }
}
```

## Detach

Отвязывает графический объект.

```
void Detach()
```

### Возвращаемое значение

Нет.

### Пример:

```
//--- example for CChartObject::Detach
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- detach chart object
    object.Detach();
}
```

## ShiftObject

Перемещает графический объект.

```
bool ShiftObject(
    datetime d_time,           // приращение координаты времени
    double   d_price           // приращение координаты цены
)
```

### Параметры

*d\_time*

[in] Приращение координаты времени всех точек привязки

*d\_price*

[in] Приращение координаты цены всех точек привязки.

### Возвращаемое значение

true - в случае удачи, false - если не удалось переместить объект.

### Пример:

```
//--- example for CChartObject::ShiftObject
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    datetime     d_time;
    double      d_price;
    //--- shift chart object
    object.ShiftObject(d_time,d_price);
}
```

## ShiftPoint

Перемещает указанную точку привязки графического объекта.

```
bool ShiftPoint(
    int      point,          // номер точки
    datetime d_time,        // приращение координаты времени
    double   d_price         // приращение координаты цены
)
```

### Параметры

*point*

[in] Номер точки привязки графического объекта.

*d\_time*

[in] Приращение координаты времени указанной точки привязки.

*d\_price*

[in] Приращение координаты цены указанной точки привязки.

### Возвращаемое значение

true - в случае удачи, false - если не удалось переместить точку.

### Пример:

```
//--- example for CChartObject::ShiftPoint
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    datetime     d_time;
    double       d_price;
    //---
    if(object.NumPoints()>0)
    {
        //--- shift point of chart object
        object.ShiftPoint(0,d_time,d_price);
    }
}
```

## Time (метод Get)

Получает координату времени указанной точки привязки графического объекта.

```
datetime Time(
    int point // номер точки
) const
```

### Параметры

*point*

[in] Номер точки привязки графического объекта.

### Возвращаемое значение

Координату времени указанной точки привязки графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, либо у объекта нет указанной точки, возвращается 0.

## Time (метод Set)

Устанавливает координату времени указанной точки привязки графического объекта.

```
bool Time(
    int point, // номер точки
    datetime new_time // время
)
```

### Параметры

*point*

[in] Номер точки привязки графического объекта.

*new\_time*

[in] Новое значение координаты времени указанной точки привязки графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить координату времени.

### Пример:

```
//--- example for CChartObject::Time
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //---
    for(int i=0;i<object.NumPoints();i++)
    {
        //--- get time of the chart object point
        datetime point_time=object.Time(i);
```

```
if(point_time==0)
{
//--- set time of the chart object point
object.Time(i,TimeCurrent());
}
}
```

## Price (метод Get)

Получает координату цены указанной точки привязки графического объекта.

```
double Price(
    int point      // номер точки
) const
```

### Параметры

*point*  
 [in] Номер точки привязки графического объекта.

### Возвращаемое значение

Координата цены указанной точки привязки графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, либо у объекта нет указанной точки, возвращается EMPTY\_VALUE.

## Price (метод Set)

Устанавливает координату цены указанной точки привязки графического объекта.

```
bool Price(
    int     point,        // номер точки
    double new_price     // цена
)
```

### Параметры

*point*  
 [in] Номер точки привязки графического объекта.  
*new\_price*  
 [in] Новое значение координаты цены указанной точки привязки графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить координату цены.

### Пример:

```
//--- example for CChartObject::Price
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    double       price;
//---
for(int i=0;i<object.NumPoints();i++)
{
//--- get price of the chart object point
```

```
double point_price=object.Price(i);
if (point_price!=price)
{
    //--- set price for the chart object point
    object.Price(i,price);
}
}
```

## Color (метод Get)

Получает цвет линии графического объекта.

```
color Color() const
```

### Возвращаемое значение

Цвет линии графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта возвращается CLR\_NONE.

## Color (метод Set)

Устанавливает цвет линии графического объекта.

```
bool Color(
    color new_color // новый цвет
)
```

### Параметры

*new\_color*  
 [in] Новое значение цвета линии графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

### Пример:

```
//--- example for CChartObject::Color
#include <ChartObjects\ChartObject.mqh>
//---

void OnStart()
{
    CChartObject object;
    //--- get color of chart object
    color object_color=object.Color();
    if(object_color!=clrRed)
    {
        //--- set color of chart object
        object.Color(clrRed);
    }
}
```

## Style (метод Get)

Получает стиль линии графического объекта.

```
ENUM_LINE_STYLE Style() const
```

### Возвращаемое значение

Стиль линии графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается `WRONG_VALUE`.

## Style (метод Set)

Устанавливает стиль линии графического объекта.

```
bool Style(
    ENUM_LINE_STYLE new_style      // стиль
)
```

### Параметры

`new_style`

[in] Новое значение стиля линии графического объекта.

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить стиль.

### Пример:

```
//--- example for CChartObject::Style
#include <ChartObjects\ChartObject.mqh>
//---

void OnStart()
{
    CChartObject object;
    //--- get style of chart object
    ENUM_LINE_STYLE style=object.Style();
    if(style!=STYLE_SOLID)
    {
        //--- set style of chart object
        object.Style(STYLE_SOLID);
    }
}
```

**Смотри также**

## Width (метод Get)

Получает толщину линии графического объекта.

```
int Width() const
```

### Возвращаемое значение

Толщину линии графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается -1.

## Width (метод Set)

Устанавливает толщину линии графического объекта.

```
bool Width(
    int new_width // толщина
)
```

### Параметры

*new\_width*

[in] Новое значение толщины линии графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить толщину.

### Пример:

```
//--- example for CChartObject::Width
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get width of chart object
    int width=object.Width();
    if(width!=1)
    {
        //--- set width of chart object
        object.Width(1);
    }
}
```

## Background (метод Get)

Получает флаг рисования графического объекта на заднем плане.

```
bool Background() const
```

### Возвращаемое значение

Флаг рисования на заднем плане, графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта возвращается false.

## Background (метод Set)

Устанавливает флаг рисования графического объекта на заднем плане.

```
bool Background(
    bool background // значение флага
)
```

### Параметры

*background*

[in] Новое значение флага рисования графического объекта на заднем плане.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

### Пример:

```
//--- example for CChartObject::Background
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get background flag of chart object
    bool background_flag=object.Background();
    if(!background_flag)
    {
        //--- set background flag of chart object
        object.Background(true);
    }
}
```

## Selected (метод Get)

Получает флаг "выбранности" графического объекта. Иными словами - выбран графический объект или нет.

```
bool Selected() const
```

### Возвращаемое значение

Флаг "выбранности" графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается `false`.

## Selected (метод Set)

Устанавливает флаг "выбранности" графического объекта.

```
bool Selected(
    bool selected // значение флага
)
```

### Параметры

*selected*

[in] Новое значение флага "выбранности" графического объекта.

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

### Пример:

```
//--- example for CChartObject::Selected
#include <ChartObjects\ChartObject.mqh>
//---

void OnStart()
{
    CChartObject object;
    //--- get the "selected" flag of chart object
    bool selected_flag=object.Selected();
    if(selected_flag)
    {
        //--- set the "selected" flag of chart object
        object.Selected(false);
    }
}
```

## Selectable (метод Get)

Получает флаг "выбираемости" графического объекта. Иными словами - может ли графический объект быть выбран или нет.

```
bool Selectable() const
```

### Возвращаемое значение

Флаг "выбираемости" графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта возвращается `false`.

## Selectable (метод Set)

Устанавливает флаг "выбираемости" графического объекта.

```
bool Selectable(
    bool selectable           // значение флага
)
```

### Параметры

`selectable`

[in] Новое значение флага "выбираемости" графического объекта.

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

### Пример:

```
//--- example for CChartObject::Selectable
#include <ChartObjects\ChartObject.mqh>
//---

void OnStart()
{
    CChartObject object;
    //--- get the "selectable" flag of chart object
    bool selectable_flag=object.Selectable();
    if(selectable_flag)
    {
        //--- set the "selectable" flag of chart object
        object.Selectable(false);
    }
}
```

## Description (метод Get)

Получает описание (текст) графического объекта.

```
string Description() const
```

### Возвращаемое значение

Описание (текст) графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается NULL.

## Description (метод Set)

Устанавливает описание (текст) графического объекта.

```
bool Description(
    string text // текст
)
```

### Параметры

*text*

[in] Новое значение описания (текста) графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить описание (текст).

### Пример:

```
//--- example for CChartObject::Description
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get description of chart object
    string description=object.Description();
    if(description=="")
    {
        //--- set description of chart object
        object.Description("MyObject");
    }
}
```

## Tooltip (метод Get)

Получает текст всплывающей подсказки.

```
string Tooltip() const
```

### Возвращаемое значение

Текст всплывающей подсказки графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается NULL.

## Tooltip (метод Set)

Устанавливает текст всплывающей подсказки.

```
bool Tooltip(  
    string new_tooltip // новый текст всплывающей подсказки  
)
```

### Параметры

*new\_tooltip*

[in] Новое значение текста всплывающей подсказки графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить текст подсказки.

### Примечание:

Если свойство не задано, то показывается подсказка, автоматически сформированная терминалом. Показ подсказки можно отключить, установив для нее значение "\n" (перевод строки).

## Timeframes (метод Get)

Получает флаги видимости графического объекта.

```
int Timeframes() const
```

### Возвращаемое значение

Флаги видимости графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Timeframes (метод Set)

Устанавливает флаги видимости графического объекта.

```
bool Timeframes(
    int new_timeframes // флаги видимости
)
```

### Параметры

*new\_timeframes*  
 [in] Новые флаги видимости графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаги видимости.

### Пример:

```
//--- example for CChartObject::Timeframes
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get timeframes of chart object
    int timeframes=object.Timeframes();
    if(!(timeframes&OBJ_PERIOD_H1))
    {
        //--- set timeframes of chart object
        object.Timeframes(timeframes|OBJ_PERIOD_H1);
    }
}
```

## Z\_Order (метод Get)

Получает приоритет графического объекта на получение события нажатия мышки на графике ([CHARTEVENT\\_CLICK](#)).

```
long Z_Order() const
```

### Возвращаемое значение

Приоритет графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Z\_Order (метод Set)

Устанавливает приоритет графического объекта на получение события нажатия мышки на графике ([CHARTEVENT\\_CLICK](#)).

```
bool Z_Order(  
    long value // приоритет графического объекта  
)
```

### Параметры

*value*

[in] Новый приоритет графического объекта на получение события [CHARTEVENT\\_CLICK](#).

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить приоритет.

### Примечание

Свойство Z\_Order предназначено для управления приоритетом при отработке клика на графических объектах. Установив значение, больше значения по умолчанию (0), можно повысить приоритет объекта при обработке кликов мыши.

## CreateTime

Получает время создания графического объекта.

```
datetime CreateTime() const
```

### Возвращаемое значение

Время создания графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

### Пример:

```
//--- example for CChartObject::CreateTime
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get create time of chart object
    datetime create_time=object.CreateTime();
}
```

## LevelsCount (метод Get)

Получает количество уровней графического объекта.

```
int LevelsCount() const
```

### Возвращаемое значение

Количество уровней графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## LevelsCount (метод Set)

Устанавливает количество уровней графического объекта.

```
bool LevelsCount(
    int levels // количество уровней
)
```

### Параметры

*levels*

[in] Новое количество уровней графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить количество уровней.

### Пример:

```
//--- example for CChartObject::LevelsCount
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- get levels count of chart object
    int levels_count=object.LevelsCount();
    //--- set levels count of chart object
    object.LevelsCount(levels_count+1);
}
```

## LevelColor (метод Get)

Получает цвет линии указанного уровня графического объекта.

```
color LevelColor(
    int level      // номер уровня
) const
```

### Параметры

*level*

[in] Номер уровня графического объекта.

### Возвращаемое значение

Цвет линии указанного уровня графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, либо у объекта нет указанного уровня, возвращается CLR\_NONE.

## LevelColor (метод Set)

Устанавливает цвет линии указанного уровня графического объекта.

```
bool LevelColor(
    int level,          // номер уровня
    color new_color    // новый цвет
)
```

### Параметры

*level*

[in] Номер уровня графического объекта.

*new\_color*

[in] Новое значение цвета линии указанного уровня графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

### Пример:

```
//--- example for CChartObject::LevelColor
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //---
    for(int i=0;i<object.LevelsCount();i++)
    {
        //--- get level color of chart object
        color level_color=object.LevelColor(i);
        if(level_color!=clrRed)
```

```
{  
    //--- set level color of chart object  
    object.LevelColor(i,clrRed);  
}  
}  
}
```

## LevelStyle (метод Get)

Получает стиль линии указанного уровня графического объекта.

```
ENUM_LINE_STYLE LevelStyle(
    int level           // номер уровня
) const
```

### Параметры

*level*

[in] Номер уровня графического объекта.

### Возвращаемое значение

Стиль линии указанного уровня графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, либо у объекта нет указанного уровня, возвращается WRONG\_VALUE.

## LevelStyle (метод Set)

Устанавливает стиль линии указанного уровня графического объекта.

```
int LevelStyle(
    int          level,      // номер уровня
    ENUM_LINE_STYLE style     // стиль линии
)
```

### Параметры

*level*

[in] Номер уровня графического объекта.

*style*

[in] Новое значение стиля линии указанного уровня графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить стиль.

### Пример:

```
//--- example for CChartObject::LevelStyle
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //---
    for(int i=0;i<object.LevelsCount();i++)
    {
        //--- get level style of chart object
        ENUM_LINE_STYLE level_style=object.LevelStyle(i);
        if(level_style!=STYLE_SOLID)
```

```
{  
    //--- set level style of chart object  
    object.LevelStyle(i, STYLE_SOLID);  
}  
}  
}
```

## LevelWidth (метод Get)

Получает толщину линии указанного уровня графического объекта.

```
int LevelWidth(
    int level        // номер уровня
) const
```

### Параметры

*level*

[in] Номер уровня графического объекта.

### Возвращаемое значение

Толщину линии указанного уровня графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, либо у объекта нет указанного уровня, возвращается -1.

## LevelWidth (метод Set)

Устанавливает толщину линии указанного уровня графического объекта.

```
bool LevelWidth(
    int level,           // номер уровня
    int new_width        // новая толщина
)
```

### Параметры

*level*

[in] Номер уровня графического объекта.

*new\_width*

[in] Новое значение толщины линии указанного уровня графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить толщину

### Пример:

```
//--- example for CChartObject::LevelWidth
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //---
    for(int i=0;i<object.LevelsCount();i++)
    {
        //--- get level width of chart object
        int level_width=object.LevelWidth(i);
        if(level_width!=1)
```

```
{  
    //--- set level width of chart object  
    object.LevelWidth(i,1);  
}  
}  
}
```

## LevelValue (метод Get)

Получает значение указанного уровня графического объекта.

```
double LevelValue(
    int level           // номер уровня
) const
```

### Параметры

*level*

[in] Номер уровня графического объекта.

### Возвращаемое значение

Значение указанного уровня графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, либо у объекта нет указанного уровня, возвращается EMPTY\_VALUE.

## LevelValue (метод Set)

Устанавливает значение указанного уровня графического объекта.

```
bool LevelValue(
    int     level,           // номер уровня
    double new_value        // новое значение
)
```

### Параметры

*level*

[in] Номер уровня графического объекта.

*new\_value*

[in] Новое значение указанного уровня графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить значение уровня.

### Пример:

```
//--- example for CChartObject::LevelValue
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //---
    for(int i=0;i<object.LevelsCount();i++)
    {
        //--- get level value of chart object
        double level_value=object.LevelValue(i);
        if(level_value!=0.1*i)
```

```
{  
    //--- set level value of chart object  
    object.LevelValue(i,0.1*i);  
}  
}  
}
```

## LevelDescription (метод Get)

Получает описание (текст) указанного уровня графического объекта.

```
string LevelDescription(
    int level           // номер уровня
) const
```

### Параметры

*level*

[in] Номер уровня графического объекта.

### Возвращаемое значение

Описание (текст) указанного уровня графического объекта, привязанного к экземпляру класса. Если нет привязанного объекта, либо у объекта нет указанного уровня, возвращается NULL.

## LevelDescription (метод Set)

Устанавливает описание (текст) указанного уровня графического объекта.

```
bool LevelDescription(
    int level,           // номер уровня
    string text          // текст
)
```

### Параметры

*level*

[in] Номер уровня графического объекта.

*text*

[in] Новое значение описания (текста) указанного уровня графического объекта.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить описание (текст).

### Пример:

```
//--- example for CChartObject::LevelDescription
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //---
    for(int i=0;i<object.LevelsCount();i++)
    {
        //--- get level description of chart object
        string level_description=object.LevelDescription(i);
        if(level_description=="")
```

```
{  
    //--- set level description of chart object  
    object.LevelDescription(i, "Level_"+IntegerToString(i));  
}  
}  
}
```

## GetInteger

Обеспечивает упрощенный доступ к функциям API MQL5 [ObjectGetInteger\(\)](#) для получения значений integer-свойств (имеющих типы bool, char, uchar, short, ushort, int, uint, long, ulong, datetime, color) привязанного к экземпляру класса графического объекта. Существуют два варианта вызова функции:

### Получение значения свойства без проверки корректности

```
long GetInteger(
    ENUM_OBJECT_PROPERTY_INTEGER prop_id,           // идентификатор integer-свойства
    int                         modifier=-1          // модификатор
) const
```

#### Параметры

*prop\_id*  
 [in] Идентификатор double-свойства графического объекта.  
*modifier=-1*  
 [in] Модификатор (индекс) double-свойства.

#### Возвращаемое значение

Значение integer-свойства - в случае удачи, 0 - если нет возможности получить integer-свойство.

### Получение значения свойства с проверкой корректности такого обращения

```
bool GetInteger(
    ENUM_OBJECT_PROPERTY_INTEGER prop_id,           // идентификатор integer-свойства
    int                         modifier,            // модификатор
    long&                      value                // ссылка на переменную
) const
```

#### Параметры

*prop\_id*  
 [in] Идентификатор integer-свойства графического объекта.  
*modifier*  
 [in] Модификатор (индекс) integer-свойства.  
*value*  
 [out] Ссылка на переменную для размещения значения integer-свойства.

#### Возвращаемое значение

true - в случае удачи, false - если нет возможности получить integer-свойство.

#### Пример:

```
//--- example for CChartObject::GetInteger
#include <ChartObjects\ChartObject.mqh>
//---
```

```
void OnStart()
{
    CChartObject object;
    //--- get color of chart object by easy method
    printf("Objects color is %s",ColorToString(object.GetInteger(OBJPROP_COLOR),true));
    //--- get color of chart object by classic method
    long color_value;
    if(!object.GetInteger(OBJPROP_COLOR,0,color_value))
    {
        printf("Get integer property error %d",GetLastError());
        return;
    }
    else
        printf("Objects color is %s",color_value);
    for(int i=0;i<object.LevelsCount();i++)
    {
        //--- get levels width by easy method
        printf("Level %d width is %d",i,object.GetInteger(OBJPROP_LEVELWIDTH,i));
        //--- get levels width by classic method
        long width_value;
        if(!object.GetInteger(OBJPROP_LEVELWIDTH,i,width_value))
        {
            printf("Get integer property error %d",GetLastError());
            return;
        }
        else
            printf("Level %d width is %d",i,width_value);
    }
}
```

## SetInteger

Обеспечивает упрощенный доступ к функциям API MQL5 [ObjectSetInteger\(\)](#) для изменения integer-свойств (имеющих типы bool, char, uchar, short, ushort, int, uint, long, ulong, datetime, color) привязанного к экземпляру класса графического объекта. Существуют два варианта вызова функции:

### Установка значения свойства, не требующего модификатора

```
bool SetInteger(
    ENUM_OBJECT_PROPERTY_INTEGER prop_id,      // идентификатор integer-свойства
    long                         value        // значение
)
```

#### Параметры

*prop\_id*

[in] Идентификатор integer-свойства графического объекта.

*value*

[in] Новое значение изменяемого integer-свойства.

### Установка значения свойства с указанием модификатора

```
bool SetInteger(
    ENUM_OBJECT_PROPERTY_INTEGER prop_id,      // идентификатор integer-свойства
    int                          modifier,     // модификатор
    long                         value        // значение
)
```

#### Параметры

*prop\_id*

[in] Идентификатор integer-свойства графического объекта.

*modifier*

[in] Модификатор (индекс) integer-свойства.

*value*

[in] Новое значение изменяемого integer-свойства.

#### Возвращаемое значение

true - в случае удачи, false - если нет возможности изменить integer-свойство.

#### Пример:

```
//--- example for CChartObject::SetInteger
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
```

```
//--- set new color of chart object
if(!object.SetInteger(OBJPROP_COLOR,clrRed))
{
    printf("Set integer property error %d",GetLastError());
    return;
}
for(int i=0;i<object.LevelsCount();i++)
{
    //--- set levels width
    if(!object.SetInteger(OBJPROP_LEVELWIDTH,i,i))
    {
        printf("Set integer property error %d",GetLastError());
        return;
    }
}
}
```

## GetDouble

Обеспечивает упрощенный доступ к функциям API MQL5 [ObjectGetDouble\(\)](#) для получения значений double-свойств (имеющих типы float и double) привязанного к экземпляру класса графического объекта. Существуют два варианта вызова функции:

### Получение значения свойства без проверки корректности

```
double GetDouble(
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id,           // идентификатор double-свойства
    int                         modifier=-1         // модификатор
) const
```

#### Параметры

*prop\_id*

[in] Идентификатор double-свойства графического объекта.

*modifier=-1*

[in] Модификатор (индекс) double-свойства.

#### Возвращаемое значение

Значение double-свойства - в случае удачи, `EMPTY_VALUE` - если нет возможности получить double-свойство.

### Получение значения свойства с проверкой корректности такого обращения

```
bool GetDouble(
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id,           // идентификатор double-свойства
    int                         modifier,           // модификатор
    double&                     value             // ссылка на переменную
) const
```

#### Параметры

*prop\_id*

[in] Идентификатор double-свойства графического объекта.

*modifier*

[in] Модификатор (индекс) double-свойства.

*value*

[out] Ссылка на переменную для размещения значения double-свойства.

#### Возвращаемое значение

`true` - в случае удачи, `false` - если нет возможности получить double-свойство.

#### Пример:

```
//--- example for CChartObject::GetDouble
#include <ChartObjects\ChartObject.mqh>
//---
```

```
void OnStart()
{
    CChartObject object;
    //---
    for(int i=0;i<object.LevelsCount();i++)
    {
        //--- get levels value by easy method
        printf("Level %d value=%f",i,object.GetDouble(OBJPROP_LEVELVALUE,i));
        //--- get levels value by classic method
        double value;
        if(!object.SetDouble(OBJPROP_LEVELVALUE,i,value))
        {
            printf("Get double property error %d",GetLastError());
            return;
        }
        else
            printf("Level %d value=%f",i,value);
    }
}
```

## SetDouble

Обеспечивает упрощенный доступ к функциям API MQL5 [ObjectSetDouble\(\)](#) для изменения double-свойств (имеющих типы float и double) привязанного к экземпляру класса графического объекта. Существуют два варианта вызова функции:

### Установка значения свойства, не требующего модификатора

```
bool SetDouble(
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id,      // идентификатор double-свойства
    double                      value        // значение
)
```

#### Параметры

*prop\_id*  
 [in] Идентификатор double-свойства графического объекта.

*value*  
 [in] Новое значение изменяемого double-свойства.

### Установка значения свойства с указанием модификатора

```
bool SetDouble(
    ENUM_OBJECT_PROPERTY_DOUBLE prop_id,      // идентификатор double-свойства
    int                         modifier,     // модификатор
    double                      value        // значение
)
```

#### Параметры

*prop\_id*  
 [in] Идентификатор double-свойства графического объекта.

*modifier*  
 [in] Модификатор (индекс) double-свойства.

*value*  
 [in] Новое значение изменяемого double-свойства.

#### Возвращаемое значение

true - в случае удачи, false - если нет возможности изменить double-свойство.

#### Пример:

```
//--- example for CChartObject::SetDouble
#include <ChartObjects\ChartObject.mqh>
//---

void OnStart()
{
    CChartObject object;
    //---
```

```
for(int i=0;i<object.LevelsCount();i++)
{
//--- set level value of chart object
if(!object.SetDouble(OBJPROP_LEVELVALUE,i,0.1*i))
{
printf("Set double property error %d",GetLastError());
return;
}
}
```

## GetString

Обеспечивает упрощенный доступ к функциям API MQL5 [ObjectGetString\(\)](#) для получения значений string-свойств, привязанного к экземпляру класса графического объекта. Существуют два варианта вызова функции:

### Получение значения свойства без проверки корректности

```
string GetString(
    ENUM_OBJECT_PROPERTY_STRING prop_id,           // идентификатор string-свойства
    int                         modifier=-1         // модификатор
) const
```

#### Параметры

*prop\_id*

[in] Идентификатор string-свойства графического объекта.

*modifier=-1*

[in] Модификатор (индекс) string-свойства.

#### Возвращаемое значение

Значение string-свойства - в случае удачи, "" - если нет возможности получить string-свойство.

### Получение значения свойства с проверкой корректности такого обращения

```
bool GetString(
    ENUM_OBJECT_PROPERTY_STRING prop_id,           // идентификатор string-свойства
    int                         modifier,          // модификатор
    string&                     value            // ссылка на переменную
) const
```

#### Параметры

*prop\_id*

[in] Идентификатор string-свойства графического объекта.

*modifier*

[in] Модификатор (индекс) string-свойства.

*value*

[out] Ссылка на переменную для размещения значения string-свойства.

#### Возвращаемое значение

true - в случае удачи, false - если нет возможности получить string-свойство.

#### Пример:

```
//--- example for CChartObject::GetString
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
```

```
{  
    CChartObject object;  
    string      value;  
    //--- get name of chart object by easy method  
    printf("Object name is '%s'",object.GetString(OBJPROP_NAME));  
    //--- get name of chart object by classic method  
    if(!object.GetString(OBJPROP_NAME,0,value))  
    {  
        printf("Get string property error %d",GetLastError());  
        return;  
    }  
    else  
        printf("Object name is '%s'",value);  
    for(int i=0;i<object.LevelsCount();i++)  
    {  
        //--- get levels description by easy method  
        printf("Level %d description is '%s'",i,object.GetString(OBJPROP_LEVELTEXT,i));  
        //--- get levels description by classic method  
        if(!object.GetString(OBJPROP_LEVELTEXT,i,value))  
        {  
            printf("Get string property error %d",GetLastError());  
            return;  
        }  
        else  
            printf("Level %d description is '%s'",i,value);  
    }  
}
```

## SetString

Обеспечивает упрощенный доступ к функциям API MQL5 [ObjectSetString\(\)](#) для изменения string-свойств, привязанного к экземпляру класса графического объекта. Существуют два варианта вызова функции:

### Установка значения свойства, не требующего модификатора

```
bool SetString(
    ENUM_OBJECT_PROPERTY_STRING prop_id,      // идентификатор string-свойства
    string                      value        // значение
)
```

#### Параметры

*prop\_id*

[in] Идентификатор string-свойства графического объекта.

*value*

[in] Новое значение изменяемого string-свойства.

### Установка значения свойства с указанием модификатора

```
bool SetString(
    ENUM_OBJECT_PROPERTY_STRING prop_id,      // идентификатор string-свойства
    int                         modifier,       // модификатор
    string                      value        // значение
)
```

#### Параметры

*prop\_id*

[in] Идентификатор string-свойства графического объекта.

*modifier*

[in] Модификатор (индекс) string-свойства.

*value*

[in] Новое значение изменяемого string-свойства.

#### Возвращаемое значение

true - в случае удачи, false - если нет возможности изменить string-свойство.

#### Пример:

```
//--- example for CChartObject::SetString
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    CChartObject object;
    //--- set new name of chart object
```

```
if(!object.SetString(OBJPROP_NAME, "MyObject"))
{
    printf("Set string property error %d", GetLastError());
    return;
}
for(int i=0;i<object.LevelsCount();i++)
{
    //--- set levels description
    if(!object.SetString(OBJPROP_LEVELTEXT,i,"Level_"+IntegerToString(i)))
    {
        printf("Set string property error %d", GetLastError());
        return;
    }
}
}
```

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*  
 [in] хэндл ранее открытого при помощи функции FileOpen(...) файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CChartObject::Save
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
{
    int         file_handle;
    CChartObject object=new CChartObject;
    //--- set object parameters
    //--- . . .
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!object.Save(file_handle))
        {
            //--- file save error
            printf("File save: Error %d!",GetLastError());
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
}
```

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого при помощи функции FileOpen(...) файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CChartObject::Load
#include <ChartObjects\ChartObject.mqh>
//---

void OnStart()
{
    int         file_handle;
    CChartObject object;
    //--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!object.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
    //--- use object
    //--- . . .
}
```

## Type

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObject](#) - 0x8888).

### Пример:

```
//--- example for CChartObject::Type
#include <ChartObjects\ChartObject.mqh>
//---

void OnStart()
{
    CChartObject object;
    //--- get object type
    int type=object.Type();
}
```

## Объекты "Линии"

Группа графических объектов "Линии".

Этот раздел содержит технические детали работы с группой классов графических объектов "Линии" и описание соответствующих компонентов стандартной библиотеки MQL5.

Имя класса	Объект
<a href="#">CChartObjectVLine</a>	Графический объект "Вертикальная линия"
<a href="#">CChartObjectHLine</a>	Графический объект "Горизонтальная линия"
<a href="#">CChartObjectTrend</a>	Графический объект "Трендовая линия"
<a href="#">CChartObjectTrendByAngle</a>	Графический объект "Трендовая линия по углу"
<a href="#">CChartObjectCycles</a>	Графический объект "Циклические линии"

Смотри также

[Типы объектов](#), [Графические объекты](#)

## CChartObjectVLine

Класс CChartObjectVLine является классом для упрощенного доступа к свойствам графического объекта "Вертикальная линия".

### Описание

Класс CChartObjectVLine обеспечивает доступ к свойствам объекта "Вертикальная линия".

### Декларация

```
class CChartObjectVLine : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectVLine
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Вертикальная линия"
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Вертикальная линия".

```
bool Create(
    long      chart_id,      // идентификатор чарта
    string    name,          // имя объекта
    int       window,        // окно чарта
    datetime  time          // координата времени
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time*

[in] Координата времени точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Type

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectVLine](#) - OBJ\_VLINE).

## CChartObjectHLine

Класс CChartObjectHLine является классом для упрощенного доступа к свойствам графического объекта "Горизонтальная линия".

### Описание

Класс CChartObjectHLine обеспечивает доступ к свойствам объекта "Горизонтальная линия".

### Декларация

```
class CChartObjectHLine : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectHLine
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Горизонтальная линия"
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Горизонтальная линия".

```
bool Create(
    long    chart_id,      // идентификатор чарта
    string  name,         // имя объекта
    long    window,        // окно чарта
    double  price          // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*price*

[in] Координата цены точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectHLine](#) - OBJ\_HLINE).

## CChartObjectTrend

Класс CChartObjectTrend является классом для упрощенного доступа к свойствам графического объекта "Трендовая линия".

### Описание

Класс CChartObjectTrend обеспечивает доступ к свойствам объекта "Трендовая линия".

### Декларация

```
class CChartObjectTrend : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
```

### Прямые потомки

```
CChartObjectChannel, CChartObjectFibo, CChartObjectFiboChannel, CChartObjectFiboExpansion,
CChartObjectGannFan, CChartObjectGannGrid, CChartObjectPitchfork, CChartObjectRegression,
CChartObjectStdDevChannel, CChartObjectTrendByAngle
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Трендовая линия"
Свойства	
<a href="#">RayLeft</a>	Получить/установить свойство "луч влево"
<a href="#">RayRight</a>	Получить/установить свойство "луч вправо"
Ввод/вывод	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Трендовая линия".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    price2           // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

### Возвращаемое значение

`true` - в случае успешного завершения, `false` - в случае ошибки.

## RayLeft (метод Get)

Получает значение флага "Луч влево".

```
bool RayLeft() const
```

### Возвращаемое значение

Значение флага "Луч влево" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается `false`.

## RayLeft (метод Set)

Устанавливает значение флага "Луч влево".

```
bool RayLeft(  
    bool ray        // значение флага  
)
```

### Параметры

*ray*  
[in] Новое значение флага "Луч влево".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## RayRight (метод Get)

Получает значение флага "Луч вправо".

```
bool RayRight() const
```

### Возвращаемое значение

Значение флага "Луч вправо" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается false.

## RayRight (метод Set)

Устанавливает значение флага "Луч вправо".

```
bool RayRight(  
    bool ray        // значение флага  
)
```

### Параметры

*ray*  
[in] Новое значение флага "Луч влево".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого при помощи функции FileOpen(...) файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого при помощи функции FileOpen(...) файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectTrend](#) - OBJ\_TREND).

## CChartObjectTrendByAngle

Класс CChartObjectTrendByAngle является классом для упрощенного доступа к свойствам графического объекта "Трендовая линия по углу".

### Описание

Класс CChartObjectTrendByAngle обеспечивает доступ к свойствам объекта "Трендовая линия по углу".

### Декларация

```
class CChartObjectTrendByAngle : public CChartObjectTrend
```

### Заголовок

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectTrendByAngle
```

### Прямые потомки

[CChartObjectGannLine](#)

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Трендовая линия по углу"
Свойства	
<a href="#">Angle</a>	Получить/установить свойство "угол"
Ввод/вывод	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#),

[\\_LevelStyle](#), [\\_LevelStyle](#), [\\_LevelWidth](#), [\\_LevelWidth](#), [\\_LevelValue](#), [\\_LevelValue](#), [\\_LevelDescription](#),  
[\\_LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#),  
[SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [GetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Трендовая линия по углу".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    long      window,         // окно чарта
    datetime time1,          // координата времени
    double   price1,          // координата цены
    datetime time2,          // координата времени
    double   price2           // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Angle (метод Get)

Получает значение свойства "Угол".

```
double Angle() const
```

### Возвращаемое значение

Значение свойства "Угол" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается EMPTY\_VALUE.

## Angle (метод Set)

Устанавливает значение свойства "Угол".

```
bool Angle(  
    double angle // угол  
)
```

### Параметры

*angle*  
[in] Новое значение свойства "Угол".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectTrendByAngle](#) - OBJ\_TREND\_BY\_ANGLE).

## CChartObjectCycles

Класс CChartObjectCycles является классом для упрощенного доступа к свойствам графического объекта "Циклические линии".

### Описание

Класс CChartObjectCycles обеспечивает доступ к свойствам объекта "Циклические линии".

### Декларация

```
class CChartObjectCycles : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectCycles
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Циклические линии"
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Циклические линии".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    long      window,         // окно чарта
    datetime time1,          // координата времени
    double   price1,          // координата цены
    datetime time2,          // координата времени
    double   price2           // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectCycles](#) - OBJ\_CYCLES).

## Объекты "Каналы"

Группа графических объектов "Каналы".

Этот раздел содержит технические детали работы с группой классов графических объектов "Каналы" и описание соответствующих компонентов стандартной библиотеки MQL5.

Имя класса	Объект
<a href="#">CChartObjectChannel</a>	Графический объект "Равноудаленный канал"
<a href="#">CChartObjectRegression</a>	Графический объект "Канал линейной регрессии"
<a href="#">CChartObjectStdDevChannel</a>	Графический объект "Канал стандартных отклонений"
<a href="#">CChartObjectPitchfork</a>	Графический объект "Вилы Эндрюса"

Смотри также

[Типы объектов](#), [Графические объекты](#)

## CChartObjectChannel

Класс CChartObjectChannel является классом для упрощенного доступа к свойствам графического объекта "Равноудаленный канал".

### Описание

Класс CChartObjectChannel обеспечивает доступ к свойствам объекта "Равноудаленный канал".

### Декларация

```
class CChartObjectChannel : public CChartObjectTrend
```

### Заголовок

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectChannel
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Равноудаленный канал"
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Равноудаленный канал".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени первой точки привязки
    double    price1,          // координата цены первой точки привязки
    datetime  time2,          // координата времени второй точки привязки
    double    price2,          // координата цены второй точки привязки
    datetime  time3,          // координата времени третьей точки привязки
    double    price3           // координата цены третьей точки привязки
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

*time3*

[in] Координата времени третьей точки привязки.

*price3*

[in] Координата цены третьей точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectChannel](#) - OBJ\_CHANNEL).

## CChartObjectRegression

Класс CChartObjectRegression является классом для упрощенного доступа к свойствам графического объекта "Канал линейной регрессии".

### Описание

Класс CChartObjectRegression обеспечивает доступ к свойствам объекта "Канал линейной регрессии".

### Декларация

```
class CChartObjectRegression : public CChartObjectTrend
```

### Заголовок

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectRegression
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Канал линейной регрессии"
Ввод/вывод	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Канал линейной регрессии".

```
bool Create(
    long    chart_id,      // идентификатор чарта
    string  name,          // имя объекта
    long    window,         // окно чарта
    datetime time1,        // координата времени
    datetime time2          // координата времени
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectRegression](#) - OBJ\_REGRESSION).

## CChartObjectStdDevChannel

Класс CChartObjectStdDevChannel является классом для упрощенного доступа к свойствам графического объекта "Канал стандартных отклонений".

### Описание

Класс CChartObjectStdDevChannel обеспечивает доступ к свойствам объекта "Канал стандартных отклонений".

### Декларация

```
class CChartObjectStdDevChannel : public CChartObjectTrend
```

### Заголовок

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectStdDevChannel
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Канал стандартных отклонений"
Свойства	
<a href="#">Deviations</a>	Получает/устанавливает свойство "Отклонение"
Ввод/вывод	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#),

[LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),  
[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#),  
[SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [GetString](#), [ShiftObject](#), [ShiftPoint](#)

Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#)

Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Канал стандартных отклонений".

```
bool Create(
    long      chart_id,           // идентификатор чарта
    string    name,              // имя объекта
    int       window,             // окно чарта
    datetime  time1,              // координата времени
    datetime  time2,              // координата времени
    double    deviation          // отклонение
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*deviation*

[in] Значение свойства "Отклонение".

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Deviation (метод Get)

Получает значение свойства "Отклонение".

```
double Deviation() const
```

### Возвращаемое значение

Значение свойства "Отклонение" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается EMPTY\_VALUE.

## Deviation (метод Set)

Устанавливает значение свойства "Отклонение".

```
bool Deviation(  
    double deviation // отклонение  
)
```

### Параметры

*deviation*

[in] Новое значение свойства "Отклонение".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого при помощи функции FileOpen(...) файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого при помощи функции FileOpen(...) файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectStdDevChannel](#) - OBJ\_STDDEVCHANNEL).

## CChartObjectPitchfork

Класс CChartObjectPitchfork является классом для упрощенного доступа к свойствам графического объекта "Вилы Эндрюса".

### Описание

Класс CChartObjectPitchfork обеспечивает доступ к свойствам объекта "Вилы Эндрюса".

### Декларация

```
class CChartObjectPitchfork : public CChartObjectTrend
```

### Заголовок

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectPitchfork
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Вилы Эндрюса"
Ввод/вывод	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Вилы Эндрюса".

```
bool Create(
    long      chart_id,           // идентификатор чарта
    string    name,              // имя объекта
    long      window,             // окно чарта
    datetime  time1,              // координата времени первой точки привязки
    double    price1,             // координата цены первой точки привязки
    datetime  time2,              // координата времени второй точки привязки
    double    price2,             // координата цены второй точки привязки
    datetime  time3,              // координата времени третьей точки привязки
    double    price3              // координата цены третьей точки привязки
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

*time3*

[in] Координата времени третьей точки привязки.

*price3*

[in] Координата цены третьей точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectPitchfork](#) - OBJ\_PITCHFORK).

## Инструменты Ганна

Группа графических объектов "Инструменты Ганна".

Этот раздел содержит технические детали работы с группой классов графических объектов "Инструменты Ганна" и описание соответствующих компонентов стандартной библиотеки MQL5.

Имя класса	Объект
<a href="#">CChartObjectGannLine</a>	Графический объект "Линия Ганна"
<a href="#">CChartObjectGannFan</a>	Графический объект "Веер Ганна"
<a href="#">CChartObjectGannGrid</a>	Графический объект "Сетка Ганна"

Смотри также

[Типы объектов](#), [Графические объекты](#)

## CChartObjectGannLine

Класс CChartObjectGannLine является классом для упрощенного доступа к свойствам графического объекта "Линия Ганна".

### Описание

Класс CChartObjectGannLine обеспечивает доступ к свойствам объекта "Линия Ганна".

### Декларация

```
class CChartObjectGannLine : public CChartObjectTrendByAngle
```

### Заголовок

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectTrendByAngle
        CChartObjectGannLine
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Линия Ганна"
Свойства	
<a href="#">PipsPerBar</a>	Получить/установить свойство "Масштаб"
Ввод/вывод	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),

[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#),  
[SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#)

Методы унаследованные от CChartObjectTrendByAngle

[Angle](#), [Angle](#), [Create](#)

Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Линия Ганна".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    ppb              // масштаб
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*ppb*

[in] Масштаб

### Возвращаемое значение

`true` - в случае успешного завершения, `false` - в случае ошибки.

## PipsPerBar (метод Get)

Получает значение свойства "Масштаб".

```
double PipsPerBar() const
```

### Возвращаемое значение

Значение свойства "Масштаб" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается EMPTY\_VALUE.

## PipsPerBar (метод Set)

Устанавливает значение свойства "Масштаб".

```
bool PipsPerBar(  
    double ppb           // масштаб  
)
```

### Параметры

*ppb*

[in] Новое значение свойства "Масштаб".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Type

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectGannLine](#) - OBJ\_GANNLINE).

## CChartObjectGannFan

Класс CChartObjectGannFan является классом для упрощенного доступа к свойствам графического объекта "Веер Ганна".

### Описание

Класс CChartObjectGannFan обеспечивает доступ к свойствам объекта "Веер Ганна".

### Декларация

```
class CChartObjectGannFan : public CChartObjectTrend
```

### Заголовок

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectGannFan
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Веер Ганна"
Свойства	
<a href="#">PipsPerBar</a>	Получить/установить свойство "Масштаб"
<a href="#">Downtrend</a>	Получить/установить свойство "Тренд вниз"
Ввод/вывод	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),

[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#),  
[SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#)

Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Веер Ганна".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    ppb              // масштаб
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*ppb*

[in] Масштаб

### Возвращаемое значение

`true` - в случае успешного завершения, `false` - в случае ошибки.

## PipsPerBar (метод Get)

Получает значение свойства "Масштаб".

```
double PipsPerBar() const
```

### Возвращаемое значение

Значение свойства "Масштаб" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается EMPTY\_VALUE.

## PipsPerBar (метод Set)

Устанавливает значение свойства "Масштаб".

```
bool PipsPerBar(  
    double ppb           // масштаб  
)
```

### Параметры

*ppb*  
[in] Новое значение свойства "Масштаб".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Downtrend (метод Get)

Получает значение флага "Тренд вниз".

```
bool Downtrend() const
```

### Возвращаемое значение

Значение флага Тренд вниз" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается `false`.

## Downtrend (метод Set)

Устанавливает значение флага "Тренд вниз".

```
bool Downtrend(  
    bool downtrend // значение флага  
)
```

### Параметры

*downtrend*

[in] Новое значение флага "Тренд вниз".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## Save

Сохраняет данные элемента списка в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(  
    int file_handle // хэндл файла  
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Type

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectGannFan](#) - OBJ\_GANNFAN).

## CChartObjectGannGrid

Класс CChartObjectGannGrid является классом для упрощенного доступа к свойствам графического объекта "Сетка Ганна".

### Описание

Класс CChartObjectGannGrid обеспечивает доступ к свойствам объекта "Сетка Ганна".

### Декларация

```
class CChartObjectGannGrid : public CChartObjectTrend
```

### Заголовок

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectGannGrid
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Сетка Ганна"
Свойства	
<a href="#">PipsPerBar</a>	Получить/установить свойство "Масштаб"
<a href="#">Downtrend</a>	Получить/установить свойство "Тренд вниз"
Ввод/вывод	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),

[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#),  
[SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#)

Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Сетка Ганна".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    ppb              // масштаб
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*ppb*

[in] Масштаб

### Возвращаемое значение

`true` - в случае успешного завершения, `false` - в случае ошибки.

## PipsPerBar (метод Get)

Получает значение свойства "Масштаб".

```
double PipsPerBar() const
```

### Возвращаемое значение

Значение свойства "Масштаб" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается EMPTY\_VALUE.

## PipsPerBar (метод Set)

Устанавливает значение свойства "Масштаб".

```
bool PipsPerBar(  
    double ppb           // масштаб  
)
```

### Параметры

*ppb*

[in] Новое значение свойства "Масштаб".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Downtrend (метод Get)

Получает значение флага "Тренд вниз".

```
bool Downtrend() const
```

### Возвращаемое значение

Значение флага "Тренд вниз" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается `false`.

## Downtrend (метод Set)

Устанавливает значение флага "Тренд вниз".

```
bool Downtrend(  
    bool downtrend // значение флага  
)
```

### Параметры

*downtrend*

[in] Новое значение флага "Тренд вниз".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectGannGrid](#) - OBJ\_GANNGRID).

## Инструменты Фибоначчи

Группа графических объектов "Инструменты Фибоначчи".

Этот раздел содержит технические детали работы с группой классов графических объектов "Инструменты Фибоначчи" и описание соответствующих компонентов стандартной библиотеки MQL5.

Имя класса	Объект
<a href="#">CChartObjectFibo</a>	Графический объект "Линии Фибоначчи"
<a href="#">CChartObjectFiboTimes</a>	Графический объект "Временные зоны Фибоначчи"
<a href="#">CChartObjectFiboFan</a>	Графический объект "Веер Фибоначчи"
<a href="#">CChartObjectFiboArc</a>	Графический объект "Дуги Фибоначчи"
<a href="#">CChartObjectFiboChannel</a>	Графический объект "Канал Фибоначчи"
<a href="#">CChartObjectFiboExpansion</a>	Графический объект "Расширение Фибоначчи"

Смотри также

[Типы объектов](#), [Графические объекты](#)

## CChartObjectFibo

Класс CChartObjectFibo является классом для упрощенного доступа к свойствам графического объекта "Линии Фибоначчи".

### Описание

Класс CChartObjectFibo обеспечивает доступ к свойствам объекта "Линии Фибоначчи".

### Декларация

```
class CChartObjectFibo : public CChartObjectTrend
```

### Заголовок

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectFibo
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Линии Фибоначчи"
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Линии Фибоначчи".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    price2           // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectFibo](#) - OBJ\_FIBO).

## CChartObjectFiboTimes

Класс CChartObjectFiboTimes является классом для упрощенного доступа к свойствам графического объекта "Временные зоны Фибоначчи".

### Описание

Класс CChartObjectFiboTimes обеспечивает доступ к свойствам объекта "Временные зоны Фибоначчи".

### Декларация

```
class CChartObjectFiboTimes : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

### Иерархия наследования

[CObject](#)

[CChartObject](#)

CChartObjectFiboTimes

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Временные зоны Фибоначчи"
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Временные зоны Фибоначчи".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    price2           // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectFiboTimes](#) - OBJ\_FIBOTIMES).

## CChartObjectFiboFan

Класс CChartObjectFiboFan является классом для упрощенного доступа к свойствам графического объекта "Веер Фибоначчи".

### Описание

Класс CChartObjectFiboFan обеспечивает доступ к свойствам объекта "Веер Фибоначчи".

### Декларация

```
class CChartObjectFiboFan : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectFiboFan
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Веер Фибоначчи"
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Веер Фибоначчи".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    price2           // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

### Возвращаемое значение

`true` - в случае успешного завершения, `false` - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectFiboFan](#) - OBJ\_FIBOFAN).

## CChartObjectFiboArc

Класс CChartObjectFiboArc является классом для упрощенного доступа к свойствам графического объекта "Дуги Фибоначчи".

### Описание

Класс CChartObjectFiboArc обеспечивает доступ к свойствам объекта "Дуги Фибоначчи".

### Декларация

```
class CChartObjectFiboArc : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectFiboArc
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Дуги Фибоначчи"
<b>Свойства</b>	
<a href="#">Scale</a>	Получить/установить свойство "Масштаб"
<a href="#">Ellipse</a>	Получить/установить свойство "Эллипс"
<b>Ввод/вывод</b>	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#),

[LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#),  
[SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Дуги Фибоначчи".

```
bool Create(
    long      chart_id,      // идентификатор чарта
    string    name,          // имя объекта
    int       window,        // окно чарта
    datetime  time1,         // координата времени
    double    price1,        // координата цены
    datetime  time2,         // координата времени
    double    price2,        // координата цены
    double    scale          // масштаб
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

*scale*

[in] Масштаб.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Scale (метод Get)

Получает значение свойства "Масштаб".

```
double Scale() const
```

### Возвращаемое значение

Значение свойства "Масштаб" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается EMPTY\_VALUE.

## Scale (метод Set)

Устанавливает значение свойства "Масштаб".

```
bool Scale(  
    double scale        // масштаб  
)
```

### Параметры

*scale*  
[in] Новое значение свойства "Масштаб".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Ellipse (метод Get)

Получает значение флага "Эллипс".

```
bool Ellipse() const
```

### Возвращаемое значение

Значение флага "Эллипс" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается `false`.

## Ellipse (метод Set)

Устанавливает значение флага "Эллипс".

```
bool Ellipse(  
    bool ellipse // значение флага  
)
```

### Параметры

`ellipse`  
[in] Новое значение флага "Эллипс".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectFiboArc](#) - OBJ\_FIBOARC).

## CChartObjectFiboChannel

Класс CChartObjectFiboChannel является классом для упрощенного доступа к свойствам графического объекта "Канал Фибоначчи".

### Описание

Класс CChartObjectFiboChannel обеспечивает доступ к свойствам объекта "Канал Фибоначчи".

### Декларация

```
class CChartObjectFiboChannel : public CChartObjectTrend
```

### Заголовок

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectFiboChannel
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Канал Фибоначчи"
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Канал Фибоначчи".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    price2,          // координата цены
    datetime  time3,          // координата времени
    double    price3           // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

*time3*

[in] Координата времени третьей точки привязки.

*price3*

[in] Координата цены третьей точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectFiboChannel](#) - OBJ\_FIBOCHANNEL).

## CChartObjectFiboExpansion

Класс CChartObjectFiboExpansion является классом для упрощенного доступа к свойствам графического объекта "Расширение Фибоначчи".

### Описание

Класс CChartObjectFiboExpansion обеспечивает доступ к свойствам объекта "Расширение Фибоначчи".

### Декларация

```
class CChartObjectFiboExpansion : public CChartObjectTrend
```

### Заголовок

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTrend
      CChartObjectFiboExpansion
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Расширение Фибоначчи"
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Методы унаследованные от CChartObjectTrend

[RayLeft](#), [RayLeft](#), [RayRight](#), [RayRight](#), [Create](#), [Save](#), [Load](#)

Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Расширение Фибоначчи".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    price2,          // координата 2-й цены
    datetime  time3,          // координата времени
    double    price3           // координата 3-й цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

*time3*

[in] Координата времени третьей точки привязки.

*price3*

[in] Координата цены третьей точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectFiboExpansion](#) - OBJ\_EXPANSION).

## Инструменты Эллиотта

Группа графических объектов "Инструменты Эллиотта".

Этот раздел содержит технические детали работы с группой классов графических объектов "Инструменты Эллиотта".

Имя класса	Объект
<a href="#">CChartObjectElliottWave3</a>	Графический объект "Корректирующая волна"
<a href="#">CChartObjectElliottWave5</a>	Графический объект "Импульсная волна"

Смотри также

[Типы объектов](#), [Графические объекты](#)

## CChartObjectElliottWave3

Класс CChartObjectElliottWave3 является классом для упрощенного доступа к свойствам графического объекта "Корректирующая волна".

### Описание

Класс CChartObjectElliottWave3 обеспечивает доступ к свойствам объекта "Корректирующая волна".

### Декларация

```
class CChartObjectElliottWave3 : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsElliott.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectElliottWave3
```

### Прямые потомки

[CChartObjectElliottWave5](#)

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Корректирующая волна"
Свойства	
<a href="#">Degree</a>	Получить/установить свойство "Степень"
<a href="#">Lines</a>	Получить/установить свойство "Отображение линий"
Ввод/вывод	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Корректирующая волна" (3-волновка Эллиота).

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    price2,          // координата цены
    datetime  time3,          // координата времени
    double    price3           // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

*time3*

[in] Координата времени третьей точки привязки.

*price3*

[in] Координата цены третьей точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Degree (метод Get)

Получает значение свойства "Способ маркировки волн".

```
ENUM_ELLIOT_WAVE_DEGREE Degree() const
```

### Возвращаемое значение

Значение свойства "Способ маркировки волн" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается WRONG\_VALUE.

## Degree (метод Set)

Устанавливает значение свойства "Способ маркировки волн".

```
bool Degree(  
    ENUM_ELLIOT_WAVE_DEGREE degree      // значение свойства  
)
```

### Параметры

*degree*

[in] Новое значение свойства "Способ маркировки волн".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Lines (метод Get)

Получает значение флага "Отображение линий".

```
bool Lines() const
```

### Возвращаемое значение

Значение флага "Отображение линий" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается false.

## Lines (метод Set)

Устанавливает значение флага "Отображение линий".

```
bool Lines(  
    bool lines        // значение флага  
)
```

### Параметры

*lines*  
[in] Новое значение флага "Отображение линий".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## Save

Сохраняет данные элемента списка в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(  
    int file_handle // хэндл файла  
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции FileOpen(...), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectElliottWave3](#) - OBJ\_ELLIOTWAVE3).

## CChartObjectElliottWave5

Класс CChartObjectElliottWave5 является классом для упрощенного доступа к свойствам графического объекта "Импульсная волна".

### Описание

Класс CChartObjectElliottWave5 обеспечивает доступ к свойствам объекта "Импульсная волна".

### Декларация

```
class CChartObjectElliottWave5 : public CChartObjectElliottWave3
```

### Заголовок

```
#include <ChartObjects\ChartObjectsElliott.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectElliottWave3
      CChartObjectElliottWave5
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Импульсная волна"
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Методы унаследованные от CChartObjectElliottWave3

[Degree](#), [Degree](#), [Lines](#), [Lines](#), [Create](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Импульсная волна" (5-волновка Эллиота").

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    price2,          // координата цены
    datetime  time3,          // координата времени
    double    price3,          // координата цены
    datetime  time4,          // координата времени
    double    price4,          // координата цены
    datetime  time5,          // координата времени
    double    price5,          // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

*time3*

[in] Координата времени третьей точки привязки.

*price3*

[in] Координата цены третьей точки привязки.

*time4*

[in] Координата времени четвертой точки привязки.

*price4*

[in] Координата цены четвертой точки привязки.

*time5*

[in] Координата времени пятой точки привязки.

*price5*

[in] Координата цены пятой точки привязки.

#### Возвращаемое значение

`true` - в случае успешного завершения, `false` - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectElliottWave5](#) - OBJ\_ELLIOTWAVE5).

## Объекты "Фигуры"

Группа графических объектов "Фигуры".

Этот раздел содержит технические детали работы с группой классов графических объектов "Фигуры" и описание соответствующих компонентов стандартной библиотеки MQL5.

Имя класса	Объект
<a href="#">CChartObjectRectangle</a>	Графический объект "Прямоугольник"
<a href="#">CChartObjectTriangle</a>	Графический объект "Треугольник"
<a href="#">CChartObjectEllipse</a>	Графический объект "Эллипс"

Смотри также

[Типы объектов](#), [Графические объекты](#)

## CChartObjectRectangle

Класс CChartObjectRectangle является классом для упрощенного доступа к свойствам графического объекта "Прямоугольник".

### Описание

Класс CChartObjectRectangle обеспечивает доступ к свойствам объекта "Прямоугольник".

### Декларация

```
class CChartObjectRectangle : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectRectangle
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Прямоугольник"
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Прямоугольник".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    long      window,         // окно чарта
    datetime time1,          // координата времени
    double   price1,          // координата цены
    datetime time2,          // координата времени
    double   price2           // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectRectangle](#) - OBJ\_RECTANGLE).

## CChartObjectTriangle

Класс CChartObjectTriangle является классом для упрощенного доступа к свойствам графического объекта "Треугольник".

### Описание

Класс CChartObjectTriangle обеспечивает доступ к свойствам объекта "Треугольник".

### Декларация

```
class CChartObjectTriangle : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectTriangle
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Треугольник"
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Треугольник".

```
bool Create(
    long      chart_id,          // идентификатор чарта
    string    name,              // имя объекта
    long      window,            // окно чарта
    datetime  time1,             // координата времени
    double   price1,             // координата цены
    datetime  time2,             // координата времени
    double   price2,             // координата цены
    datetime  time3,             // координата времени
    double   price3              // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

*time3*

[in] Координата времени третьей точки привязки.

*price3*

[in] Координата цены третьей точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectTriangle](#) - OBJ\_TRIANGLE).

## CChartObjectEllipse

Класс CChartObjectEllipse является классом для упрощенного доступа к свойствам графического объекта "Эллипс".

### Описание

Класс CChartObjectEllipse обеспечивает доступ к свойствам объекта "Эллипс".

### Декларация

```
class CChartObjectEllipse : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectEllipse
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Эллипс"
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#), [Save](#), [Load](#)

#### Смотри также

[Типы объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Эллипс".

```
bool Create(
    long      chart_id,        // идентификатор чарта
    string    name,           // имя объекта
    int       window,          // окно чарта
    datetime  time1,          // координата времени
    double    price1,          // координата цены
    datetime  time2,          // координата времени
    double    price2,          // координата цены
    datetime  time3,          // координата времени
    double    price3           // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time1*

[in] Координата времени первой точки привязки.

*price1*

[in] Координата цены первой точки привязки.

*time2*

[in] Координата времени второй точки привязки.

*price2*

[in] Координата цены второй точки привязки.

*time3*

[in] Координата времени третьей точки привязки.

*price3*

[in] Координата цены третьей точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectEllipse](#) - OBJ\_ELLIPSE).

## Объекты "Стрелки"

Группа графических объектов "Стрелки".

Этот раздел содержит технические детали работы с группой классов графических объектов "Стрелки" и описание соответствующих компонентов стандартной библиотеки MQL5. По существу, стрелка - это некоторый значок, отображаемый пользователю, которому сопоставляется определенный код. Существуют два вида графических объектов "Стрелка" для отображения значков на графиках:

- Объект "Стрелка", который позволяет указать код значка, отображаемый объектом.
- Группа объектов для отображения определенного вида значка (и соответствующего определенному фиксированному коду).

### Класс для работы со стрелками, отображающими значки произвольного кода

Имя класса	Имя объекта-стрелки
<a href="#">CChartObjectArrow</a>	"Стрелка"

### Классы для работы со стрелками, отображающими значок фиксированного кода

Имя класса	Имя объекта-стрелки
<a href="#">CChartObjectArrowCheck</a>	"Проверка" ("Галочка")
<a href="#">CChartObjectArrowDown</a>	"Стрелка вниз"
<a href="#">CChartObjectArrowUp</a>	"Стрелка вверх"
<a href="#">CChartObjectArrowStop</a>	"Стоп"
<a href="#">CChartObjectArrowThumbUp</a>	"Хорошо" ("Большой палец вверх")
<a href="#">CChartObjectArrowThumbDown</a>	"Плохо" ("Большой палец вниз")
<a href="#">CChartObjectArrowLeftPrice</a>	Левая ценовая метка
<a href="#">CChartObjectArrowRightPrice</a>	Правая ценовая метка

#### Смотри также

[Типы объектов](#), [Способы привязки объектов](#), [Графические объекты](#)

## CChartObjectArrow

Класс CChartObjectArrow является классом для упрощенного доступа к свойствам графических объектов "Стрелки".

### Описание

Класс CChartObjectArrow обеспечивает всем своим потомкам доступ к общим свойствам объектов "Стрелки".

### Декларация

```
class CChartObjectArrow : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsArrows.mqh>
```

### Иерархия наследования

[CObject](#)

[CChartObject](#)

CChartObjectArrow

### Прямые потомки

CChartObjectArrowCheck,	CChartObjectArrowDown,	CChartObjectArrowLeftPrice,
CChartObjectArrowRightPrice,	CChartObjectArrowStop,	CChartObjectArrowThumbDown,
CChartObjectArrowThumbUp,	CChartObjectArrowUp	

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Стрелка"
Свойства	
<a href="#">ArrowCode</a>	Получает/устанавливает код символа
<a href="#">Anchor</a>	Получает/устанавливает тип привязки
Ввод/вывод	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Смотри также

[Типы объектов](#), [Способы привязки объектов](#), [Графические объекты](#)

## Create

Создает графический объект "стрелка".

```
bool Create(
    long      chart_id,      // идентификатор чарта
    string    name,          // имя объекта
    int       window,        // окно чарта
    datetime  time,          // время
    double    price,         // цена
    char      code           // код
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time*

[in] Координата времени.

*price*

[in] Координата цены.

*code*

[in] Код "стрелки" (Wingdings).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CChartObjectArrow::Create
#include <ChartObjects\ChartObjectsArrows.mqh>
//---

void OnStart()
{
    CChartObjectArrow arrow;
//--- set object parameters
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
    {
//--- arrow create error
        printf("Arrow create: Error %d!",GetLastError());
    }
}
```

```
    return;
}
//--- use arrow
//--- . . .
}
```

## ArrowCode (метод Get)

Получает код символа "стрелки".

```
char ArrowCode() const
```

### Возвращаемое значение

Код символа "стрелки" объекта, привязанного к экземпляру класса. Если нет привязанного объекта возвращается 0.

## ArrowCode (метод Set)

Устанавливает код символа "стрелки".

```
bool ArrowCode(
    char code        // значение кода
)
```

### Параметры

*code*  
 [in] Новое значение кода "стрелки" (Wingdings).

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить код.

### Пример:

```
//--- example for CChartObjectArrow::ArrowCode
#include <ChartObjects\ChartObjectsArrows.mqh>
//---

void OnStart()
{
    CChartObjectArrow arrow;
    char             code=181;
//--- set object parameters
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,code))
    {
//--- arrow create error
        printf("Arrow create: Error %d!",GetLastError());
//---
        return;
    }
//--- change the code of arrow
//--- . . .
//--- get code of arrow
    if(arrow.ArrowCode()!=code)
    {
//--- set code of arrow
```

```
    arrow.ArrowCode(code);
}
//--- use arrow
//--- . . .
}
```

## Anchor (метод Get)

Получает способ привязки объекта "стрелка" кチャрту.

```
ENUM_ARROW_ANCHOR Anchor() const
```

### Возвращаемое значение

Способ привязки объекта "стрелка", привязанного к экземпляру класса, к чарту. Если нет привязанного объекта возвращается **WRONG\_VALUE**.

## Anchor (метод Set)

Устанавливает способ привязки объекта "стрелка" кчарту.

```
bool Anchor(
    ENUM_ARROW_ANCHOR anchor        // тип привязки
)
```

### Параметры

*anchor*  
 [in] Новое значение способа привязки.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить тип привязки.

### Пример:

```
//--- example for CChartObject::Anchor
#include <ChartObjects\ChartObjectsArrows.mqh>
//---

void OnStart()
{
    CChartObjectArrow arrow;
    ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM;
//--- set object parameters
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
    {
//--- arrow create error
        printf("Arrow create: Error %d!",GetLastError());
//---
        return;
    }
//--- get anchor of arrow
    if(arrow.Anchor()!=anchor)
    {
//--- set anchor of arrow
        arrow.Anchor(anchor);
    }
}
```

```
//--- use arrow  
//--- . . .  
}
```

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*  
 [in] хэндл ранее открытого при помощи функции FileOpen(...) файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CChartObjectArrow::Save
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
{
    int             file_handle;
    CChartObjectArrow arrow;
//--- set object parameters
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
    {
//--- arrow create error
        printf("Arrow create: Error %d!",GetLastError());
//---
        return;
    }
//--- open file
    file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!arrow.Save(file_handle))
        {
//--- file save error
            printf("File save: Error %d!",GetLastError());
            FileClose(file_handle);
//---
            return;
        }
        FileClose(file_handle);
    }
}
```

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle // хэндл файла
)
```

### Параметры

*file\_handle*

[in] Хэндл ранее открытого при помощи функции FileOpen(...) файла.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CChartObjectArrow::Load
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
{
    int             file_handle;
    CChartObjectArrow arrow;
//--- open file
    file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
    if(file_handle>=0)
    {
        if(!arrow.Load(file_handle))
        {
            //--- file load error
            printf("File load: Error %d!",GetLastError());
            FileClose(file_handle);
            //---
            return;
        }
        FileClose(file_handle);
    }
//--- use arrow
//--- . . .
}
```

## Type

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectArrow](#) - OBJ\_ARROW).

### Пример:

```
//--- example for CChartObjectArrow::Type
#include <ChartObjects\ChartObjectsArrows.mqh>
//---

void OnStart()
{
    CChartObjectArrow arrow;
    //--- get arrow type
    int type=arrow.Type();
}
```

## Стрелки с фиксированным кодом

Классы "Стрелки с фиксированным кодом" являются классами для упрощенного доступа к свойствам графических объектов:

Имя класса	Имя объекта-стрелки
CChartObjectArrowCheck	"Проверка" ("Галочка")
CChartObjectArrowDown	"Стрелка вниз"
CChartObjectArrowUp	"Стрелка вверх"
CChartObjectArrowStop	"Стоп"
CChartObjectArrowThumbUp	"Хорошо" ("Большой палец вверх")
CChartObjectArrowThumbDown	"Плохо" ("Большой палец вниз")
CChartObjectArrowLeftPrice	Левая ценовая метка
CChartObjectArrowRightPrice	Правая ценовая метка

### Описание

Классы "Стрелки с фиксированным кодом" обеспечивают доступ к свойствам соответствующих объектов.

### Декларации

```
class CChartObjectArrowCheck      : public CChartObjectArrow;
class CChartObjectArrowDown     : public CChartObjectArrow;
class CChartObjectArrowUp       : public CChartObjectArrow;
class CChartObjectArrowStop     : public CChartObjectArrow;
class CChartObjectArrowThumbDown : public CChartObjectArrow;
class CChartObjectArrowThumbUp  : public CChartObjectArrow;
class CChartObjectArrowLeftPrice: public CChartObjectArrow;
class CChartObjectArrowRightPrice: public CChartObjectArrow;
```

### Заголовок

```
#include <ChartObjects\ChartObjectsArrows.mqh>
```

### Методы класса по группам

Создание	
<a href="#"><u>Create</u></a>	Создает графический объект, соответствующий классу
Свойства	
<a href="#"><u>ArrowCode</u></a>	"Заглушка" для метода изменения кода символа

Ввод/вывод	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

Смотри также

[Типы объектов](#), [Способы привязки объектов](#), [Графические объекты](#)

## Create

Создает графический объект "стрелка с фиксированным кодом".

```
bool Create(
    long      chart_id,      // идентификатор чарта
    string    name,          // имя объекта
    int       window,        // окно чарта
    datetime  time,          // время
    double    price          // цена
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time*

[in] Координата времени.

*price*

[in] Координата цены.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

### Пример:

```
//--- example for CChartObjectArrowCheck::Create
//--- example for CChartObjectArrowDown::Create
//--- example for CChartObjectArrowUp::Create
//--- example for CChartObjectArrowStop::Create
//--- example for CChartObjectArrowThumbDown::Create
//--- example for CChartObjectArrowThumbUp::Create
//--- example for CChartObjectArrowLeftPrice::Create
//--- example for CChartObjectArrowRightPrice::Create
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
{
//--- for example, take CChartObjectArrowCheck
    CChartObjectArrowCheck arrow;
//--- set object parameters
    double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
    if(!arrow.Create(0,"ArrowCheck",0,TimeCurrent(),price))
```

```
{  
    //--- arrow create error  
    printf("Arrow create: Error %d!", GetLastError());  
    //---  
    return;  
}  
//--- use arrow  
//--- . . .  
}
```

## ArrowCode

Запрещает изменение кода символа "стрелки".

```
bool ArrowCode(
    char code          // значение кода
)
```

### Параметры

*code*

[in] Любое значение.

### Возвращаемое значение

Всегда false.

### Пример:

```
//--- example for CChartObjectArrowCheck::ArrowCode
//--- example for CChartObjectArrowDown::ArrowCode
//--- example for CChartObjectArrowUp::ArrowCode
//--- example for CChartObjectArrowStop::ArrowCode
//--- example for CChartObjectArrowThumbDown::ArrowCode
//--- example for CChartObjectArrowThumbUp::ArrowCode
//--- example for CChartObjectArrowLeftPrice::ArrowCode
//--- example for CChartObjectArrowRightPrice::ArrowCode
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
{
//--- for example, take CChartObjectArrowCheck
CChartObjectArrowCheck arrow;
//--- set object parameters
double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
if(!arrow.Create(0,"ArrowCheck",0,TimeCurrent(),price))
{
//--- arrow create error
printf("Arrow create: Error %d!",GetLastError());
//---
return;
}
//--- set code of arrow
if(!arrow.ArrowCode(181))
{
//--- it is not error
printf("Arrow code can not be changed");
}
//--- use arrow
//--- . . .
}
```



## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта:

CChartObjectArrowCheck - OBJ\_ARROW\_CHECK,  
CChartObjectArrowDown - OBJ\_ARROW\_DOWN,  
CChartObjectArrowUp - OBJ\_ARROW\_UP,  
CChartObjectArrowStop - OBJ\_ARROW\_STOP,  
CChartObjectArrowThumbDown - OBJ\_ARROW\_THUMB\_DOWN,  
CChartObjectArrowThumbUp - OBJ\_ARROW\_THUMB\_UP,  
CChartObjectArrowLeftPrice - OBJ\_ARROW\_LEFT\_PRICE,  
CChartObjectArrowRightPrice - OBJ\_ARROW\_RIGHT\_PRICE.

### Пример:

```
//--- example for CChartObjectArrowCheck::Type  
//--- example for CChartObjectArrowDown::Type  
//--- example for CChartObjectArrowUp::Type  
//--- example for CChartObjectArrowStop::Type  
//--- example for CChartObjectArrowThumbDown::Type  
//--- example for CChartObjectArrowThumbUp::Type  
//--- example for CChartObjectArrowLeftPrice::Type  
//--- example for CChartObjectArrowRightPrice::Type  
#include <ChartObjects\ChartObjectsArrows.mqh>  
//---  
void OnStart()  
{  
    //--- for example, take CChartObjectArrowCheck  
    CChartObjectArrowCheck arrow;  
    //--- get arrow type  
    int type=arrow.Type();  
}
```

## Элементы управления

Группа графических объектов "Элементы управления".

Этот раздел содержит технические детали работы с группой классов графических объектов "Элементы управления" и описание соответствующих компонентов стандартной библиотеки MQL5.

Имя класса	Объект
<a href="#">CChartObjectText</a>	Графический объект "Текст"
<a href="#">CChartObjectLabel</a>	Графический объект "Текстовая метка"
<a href="#">CChartObjectEdit</a>	Графический объект "Поле ввода"
<a href="#">CChartObjectButton</a>	Графический объект "Кнопка"
<a href="#">CChartObjectSubChart</a>	Графический объект "График"
<a href="#">CChartObjectBitmap</a>	Графический объект "Рисунок"
<a href="#">CChartObjectBmpLabel</a>	Графический объект "Графическая метка"
<a href="#">CChartObjectRectLabel</a>	Графический объект "Прямоугольная метка"

Смотри также

[Типы объектов](#), [Графические объекты](#)

## CChartObjectText

Класс CChartObjectText является классом для упрощенного доступа к свойствам графического объекта "Текст".

### Описание

Класс CChartObjectText обеспечивает доступ к свойствам объекта "Текст".

### Декларация

```
class CChartObjectText : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectText
```

### Прямые потомки

[CChartTextLabel](#)

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Текст"
Свойства	
<a href="#">Angle</a>	Получить/установить свойство "Угол"
<a href="#">Font</a>	Получить/установить свойство "Шрифт"
<a href="#">FontSize</a>	Получить/установить свойство "Размер шрифта"
<a href="#">Anchor</a>	Получает/устанавливает свойство "Точка привязки"
Ввод/вывод	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

Прямые потомки класса:

- [CChartObjectLabel](#)

Смотри также

[Типы объектов](#), [Свойства объектов](#), [Способы привязки объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Текст".

```
bool Create(
    long      chart_id,      // идентификатор чарта
    string    name,          // имя объекта
    int       window,        // окно чарта
    datetime  time,          // координата времени
    double    price          // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time*

[in] Координата времени точки привязки.

*price*

[in] Координата цены точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Angle (метод Get)

Получает значение свойства "Угол наклона".

```
double Angle() const
```

### Возвращаемое значение

Значение свойства "Угол наклона" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается EMPTY\_VALUE.

## Angle (метод Set)

Устанавливает значение свойства "Угол наклона".

```
bool Angle(  
    double angle // значение свойства  
)
```

### Параметры

*angle*  
[in] Новое значение свойства "Угол наклона".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Font (метод Get)

Получает значение свойства "Шрифт".

```
string Font() const
```

### Возвращаемое значение

Значение свойства "Шрифт" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается "".

## Font (метод Set)

Устанавливает значение свойства "Шрифт".

```
bool Font(  
    string font // значение свойства  
)
```

### Параметры

*font*  
[in] Новое значение свойства "Шрифт".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## FontSize (метод Get)

Получает значение свойства "Размер шрифта".

```
int FontSize() const
```

### Возвращаемое значение

Значение свойства "Размер шрифта" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## FontSize (метод Set)

Устанавливает значение свойства "Размер шрифта".

```
bool FontSize(  
    int size // значение свойства  
)
```

### Параметры

*size*

[in] Новое значение свойства "Размер шрифта".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Anchor (метод Get)

Получает значение свойства "Положение точки привязки".

```
ENUM_ANCHOR_POINT Anchor() const
```

### Возвращаемое значение

Значение свойства "Положение точки привязки" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается `WRONG_VALUE`.

## Anchor (метод Set)

Устанавливает значение свойства "Положение точки привязки".

```
bool Anchor(  
    ENUM_ANCHOR_POINT anchor // значение свойства  
)
```

### Параметры

*anchor*

[in] Новое значение свойства "Положение точки привязки".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить свойство.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Type

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectText](#) - OBJ\_TEXT).

## CChartObjectLabel

Класс CChartObjectLabel является классом для упрощенного доступа к свойствам графического объекта "Текстовая метка".

### Описание

Класс CChartObjectLabel обеспечивает доступ к свойствам объекта "Текстовая метка".

### Декларация

```
class CChartObjectLabel : public CChartObjectText
```

### Заголовок

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectText
      CChartObjectLabel
```

### Прямые потомки

[CChartObjectEdit](#), [CChartObjectRectLabel](#)

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Текстовая метка"
Свойства	
<a href="#">X_Distance</a>	Получает/устанавливает свойство "Координата X"
<a href="#">Y_Distance</a>	Получает/устанавливает свойство "Координата Y"
<a href="#">X_Size</a>	Получает свойство "Размер по X"
<a href="#">Y_Size</a>	Получает свойство "Размер по Y"
<a href="#">Corner</a>	Получает/устанавливает свойство "Угол графика для привязки"
<a href="#">Time</a>	"Заглушка" изменения координаты времени
<a href="#">Price</a>	"Заглушка" изменения координаты цены
Ввод/вывод	

virtual <a href="#">Save</a>	Виртуальный метод записи в файл
virtual <a href="#">Load</a>	Виртуальный метод чтения из файла
virtual <a href="#">Type</a>	Виртуальный метод идентификации

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

**Методы унаследованные от CChartObject**

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

**Методы унаследованные от CChartObjectText**

[Angle](#), [Angle](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [Anchor](#), [Anchor](#), [Create](#)

**Смотри также**

[Типы объектов](#), [Свойства объектов](#), [Способы привязки объектов](#), [Угол привязки](#), [Графические объекты](#)

## Create

Создает графический объект "Текстовая метка".

```
bool Create(
    long    chart_id,      // идентификатор чарта
    string  name,         // имя объекта
    int     window,        // окно чарта
    int     X,             // координата X
    int     Y              // координата Y
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*X*

[in] Дистанция по оси X.

*Y*

[in] Дистанция по оси Y.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## X\_Distance (метод Get)

Получает значение свойства "Дистанция по оси X".

```
int X_Distance() const
```

### Возвращаемое значение

Значение свойства "Дистанция по оси X" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## X\_Distance (метод Set)

Устанавливает значение свойства "Дистанция по оси X".

```
bool X_Distance(  
    int X // значение свойства  
)
```

### Параметры

X

[in] Новое значение свойства "Дистанция по оси X".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Y\_Distance (метод Get)

Получает значение свойства "Дистанция по оси Y".

```
int Y_Distance() const
```

### Возвращаемое значение

Значение свойства "Дистанция по оси Y" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Y\_Distance (метод Set)

Устанавливает значение свойства "Дистанция по оси Y".

```
bool Y_Distance(  
    int Y // значение свойства  
)
```

### Параметры

*Y*

[in] Новое значение свойства "Дистанция по оси Y".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## X\_Size

Получает значение свойства "Размер по оси X".

```
int X_Size() const
```

### Возвращаемое значение

Значение свойства "Размер по оси X" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Y\_Size

Получает значение свойства "Размер по оси Y".

```
int Y_Size() const
```

### Возвращаемое значение

Значение свойства "Размер по оси Y" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Corner (метод Get)

Получает значение свойства "Угол графика для привязки".

```
ENUM_BASE_CORNER Corner() const
```

### Возвращаемое значение

Значение свойства "Угол графика для привязки" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается `WRONG_VALUE`.

## Corner (метод Set)

Устанавливает значение свойства "Угол графика для привязки".

```
bool Corner(  
    ENUM_BASE_CORNER corner // значение свойства  
)
```

### Параметры

*corner*  
[in] Новое значение свойства "Угол графика для привязки".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить свойство.

## Time

Запрещает изменение координаты времени.

```
bool Time(
    datetime time          // любое значение
)
```

### Параметры

*time*

[in] Любое значение типа datetime.

### Возвращаемое значение

всегда false.

## Price

Запрещает изменение координаты цены.

```
bool Price(
    double price           // любое значение
)
```

### Параметры

*price*  
[in] Любое значение типа double.

### Возвращаемое значение

всегда false.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(  
    int file_handle // хэндл файла  
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectLabel](#) - OBJ\_LABEL).

## CChartObjectEdit

Класс CChartObjectEdit является классом для упрощенного доступа к свойствам графического объекта "Поле ввода".

### Описание

Класс CChartObjectEdit обеспечивает доступ к свойствам объекта "Поле ввода".

### Декларация

```
class CChartObjectEdit : public CChartObjectLabel
```

### Заголовок

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectText
      CChartObjectLabel
        CChartObjectEdit
```

### Прямые потомки

[CChartObjectButton](#)

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Поле ввода"
Свойства	
<a href="#">TextAlign</a>	Получает/устанавливает свойство "Тип выравнивания текста"
<a href="#">X_Size</a>	Получает свойство "Размер по X"
<a href="#">Y_Size</a>	Получает свойство "Размер по Y"
<a href="#">BackColor</a>	Получает/устанавливает свойство "Цвет фона"
<a href="#">BorderColor</a>	Получает/устанавливает свойство "Цвет рамки"
<a href="#">ReadOnly</a>	Получает/устанавливает свойство "Read Only"
<a href="#">Angle</a>	"Заглушка" для изменений свойства "Угол"
Ввод/вывод	

virtual <a href="#">Save</a>	Виртуальный метод записи в файл
virtual <a href="#">Load</a>	Виртуальный метод чтения из файла
virtual <a href="#">Type</a>	Виртуальный метод идентификации

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CChartObject**

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

**Методы унаследованные от CChartObjectText**[Angle](#), [Angle](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [Anchor](#), [Anchor](#), [Create](#)**Методы унаследованные от CChartObjectLabel**

[X\\_Distance](#), [X\\_Distance](#), [Y\\_Distance](#), [Y\\_Distance](#), [X\\_Size](#), [Y\\_Size](#), [Corner](#), [Corner](#), [Time](#), [Price](#), [Create](#)

**Смотри также**

[Типы объектов](#), [Свойства объектов](#), [Способы привязки объектов](#), [Угол привязки](#), [Графические объекты](#)

## Create

Создает графический объект "Поле ввода".

```
bool Create(
    long    chart_id,      // идентификатор чарта
    string  name,         // имя объекта
    int     window,        // окно чарта
    int     X,             // координата X
    int     Y,             // координата Y
    int     sizeX,         // размер X
    int     sizeY          // размер Y
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*X*

[in] Дистанция по оси X.

*Y*

[in] Дистанция по оси Y.

*sizeX*

[in] Размер по оси X.

*sizeY*

[in] Размер по оси Y.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## .TextAlign (метод Get)

Получает значение свойства "Тип выравнивания текста по горизонтали" графического объекта "Поле ввода".

```
ENUM_ALIGN_MODE TextAlign() const
```

### Возвращаемое значение

Значение свойства "Тип выравнивания текста по горизонтали" объекта, привязанного к экземпляру класса.

## .TextAlign (метод Set)

Устанавливает значение свойства "Тип выравнивания текста по горизонтали" графического объекта "Поле ввода".

```
bool TextAlign(  
    ENUM_ALIGN_MODE align           // значение свойства  
)
```

### Параметры

*align*

[in] Новое значение свойства "Тип выравнивания текста по горизонтали".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## X\_Size

Устанавливает значение свойства "Размер по оси X".

```
bool X_Size(  
    int size // значение свойства  
)
```

### Параметры

*size*

[in] Новое значение свойства "Размер по оси X".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Y\_Size

Устанавливает значение свойства "Размер по оси Y".

```
bool Y_Size(  
    int size // значение свойства  
)
```

### Параметры

*size*

[in] Новое значение свойства "Размер по оси Y".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## BackColor (метод Get)

Получает значение свойства "Цвет фона".

```
color BackColor() const
```

### Возвращаемое значение

Значение свойства "Цвет фона" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается CLR\_NONE.

## BackColor (метод Set)

Устанавливает значение свойства "Цвет фона".

```
bool BackColor(  
    color new_color // значение свойства  
)
```

### Параметры

*new\_color*

[in] Новое значение свойства "Цвет фона".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## BorderColor (метод Get)

Получает значение свойства "Цвет рамки".

```
color BorderColor() const
```

### Возвращаемое значение

Значение свойства "Цвет рамки" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается CLR\_NONE.

## BorderColor (метод Set)

Устанавливает значение свойства "Цвет рамки".

```
bool BorderColor(  
    color new_color // новое значение свойства  
)
```

### Параметры

*new\_color*

[in] Новое значение свойства "Цвет рамки".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## ReadOnly (метод Get)

Получает значение свойства "Read Only".

```
bool ReadOnly() const
```

### Возвращаемое значение

Значение свойства "Read Only" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается false.

## ReadOnly (метод Set)

Устанавливает значение свойства "Read Only".

```
bool ReadOnly(  
    const bool flag // новое значение свойства  
)
```

### Параметры

*flag*

[in] Новое значение свойства "Read Only" (true означает запрет возможности редактирования текста).

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Angle

Запрещает изменение свойства "Угол наклона".

```
bool Angle(  
    double angle           // любое значение  
)
```

### Параметры

*angle*  
[in] Любое значение типа double.

### Возвращаемое значение

всегда false.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Type

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectEdit](#) - OBJ\_EDIT).

## CChartObjectButton

Класс CChartObjectButton является классом для упрощенного доступа к свойствам графического объекта "Кнопка".

### Описание

Класс CChartObjectButton обеспечивает доступ к свойствам объекта "Кнопка".

### Декларация

```
class CChartObjectButton : public CChartObjectEdit
```

### Заголовок

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectText
      CChartObjectLabel
      CChartObjectEdit
        CChartObjectButton
```

### Прямые потомки

CChartObjectPanel

### Методы класса по группам

Создание	
<a href="#">Create</a>	Унаследован от класса <a href="#">CChartObjectEdit</a>
Свойства	
<a href="#">State</a>	Получает/устанавливает состояние кнопки (Нажата/Отжата)
Ввод/вывод	
virtual <a href="#">Save</a>	Виртуальный метод записи в файл
virtual <a href="#">Load</a>	Виртуальный метод чтения из файла
virtual <a href="#">Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Методы унаследованные от CChartObjectText

[Angle](#), [Angle](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [Anchor](#), [Anchor](#), [Create](#)

#### Методы унаследованные от CChartObjectLabel

[X\\_Distance](#), [X\\_Distance](#), [Y\\_Distance](#), [Y\\_Distance](#), [X\\_Size](#), [Y\\_Size](#), [Corner](#), [Corner](#), [Time](#), [Price](#), [Create](#)

#### Методы унаследованные от CChartObjectEdit

[X\\_Size](#), [Y\\_Size](#), [BackColor](#), [BackColor](#), [BorderColor](#), [BorderColor](#), [ReadOnly](#), [ReadOnly](#),  [TextAlign](#),  [TextAlign](#), [Angle](#), [Create](#)

#### Смотри также

[Типы объектов](#), [Свойства объектов](#), [Способы привязки объектов](#), [Угол привязки](#), [Графические объекты](#)

## State (метод Get)

Получает значение свойства "Состояние".

```
bool State() const
```

### Возвращаемое значение

Значение свойства "Состояние" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается false.

## State (метод Set)

Устанавливает значение свойства "Состояние".

```
bool State(  
    bool state // значение свойства  
)
```

### Параметры

state

[in] Новое значение свойства "Состояние".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(  
    int file_handle // хэндл файла  
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectButton](#) - OBJ\_BUTTON).

## CChartObjectSubChart

Класс CChartObjectSubChart является классом для упрощенного доступа к свойствам графического объекта "График".

### Описание

Класс CChartObjectSubChart обеспечивает доступ к свойствам объекта "График".

### Декларация

```
class CChartObjectSubChart : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectSubChart.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectSubChart
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "График"
Свойства	
<a href="#">X_Distance</a>	Получает/устанавливает свойство "Координата X"
<a href="#">Y_Distance</a>	Получает/устанавливает свойство "Координата Y"
<a href="#">Corner</a>	Получает/устанавливает свойство "Угол графика для привязки"
<a href="#">X_Size</a>	Получает/устанавливает свойство "Размер по X"
<a href="#">Y_Size</a>	Получает/устанавливает свойство "Размер по Y"
<a href="#">Symbol</a>	Получает/устанавливает свойство "Символ"
<a href="#">Period</a>	Получает/устанавливает свойство "Период"
<a href="#">Scale</a>	Получает/устанавливает свойство "Масштаб"
<a href="#">DateScale</a>	Получает/устанавливает свойство "Отображать шкалу времени"

<a href="#"><u>PriceScale</u></a>	Получает/устанавливает "Отображать ценовую шкалу"	свойство
<a href="#"><u>Time</u></a>	"Заглушка" изменения координаты времени	
<a href="#"><u>Price</u></a>	"Заглушка" изменения координаты цены	
<a href="#"><u>Ввод/вывод</u></a>		
<a href="#"><u>virtual Save</u></a>	Виртуальный метод записи в файл	
<a href="#"><u>virtual Load</u></a>	Виртуальный метод чтения из файла	
<a href="#"><u>virtual Type</u></a>	Виртуальный метод идентификации	

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CChartObject**

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

**Смотри также**[Типы объектов](#), [Свойства объектов](#), [Угол привязки](#), [Графические объекты](#)

## Create

Создает графический объект "График на графике".

```
bool Create(
    long    chart_id,      // идентификатор чарта
    string  name,         // имя объекта
    int     window,        // окно чарта
    int     X,             // координата X
    int     Y,             // координата Y
    int     sizeX,         // размер X
    int     sizeY          // размер Y
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*X*

[in] Дистанция по оси X.

*Y*

[in] Дистанция по оси Y.

*sizeX*

[in] Размер по оси X.

*sizeY*

[in] Размер по оси Y.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## X\_Distance (метод Get)

Получает значение свойства "Дистанция по оси X".

```
int X_Distance() const
```

### Возвращаемое значение

Значение свойства "Дистанция по оси X" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## X\_Distance (метод Set)

Устанавливает значение свойства "Дистанция по оси X".

```
bool X_Distance(  
    int X // значение свойства  
)
```

### Параметры

X

[in] Новое значение свойства "Дистанция по оси X".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Y\_Distance (метод Get)

Получает значение свойства "Дистанция по оси Y".

```
int Y_Distance() const
```

### Возвращаемое значение

Значение свойства "Дистанция по оси Y" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Y\_Distance (метод Set)

Устанавливает значение свойства "Дистанция по оси Y".

```
bool Y_Distance(  
    int Y // значение свойства  
)
```

### Параметры

*Y*

[in] Новое значение свойства "Дистанция по оси Y".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Corner (метод Get)

Получает значение свойства "Угол графика для привязки".

```
ENUM_BASE_CORNER Corner() const
```

### Возвращаемое значение

Значение свойства "Угол графика для привязки" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается `WRONG_VALUE`.

## Corner (метод Set)

Устанавливает значение свойства "Угол графика для привязки".

```
bool Corner(  
    ENUM_BASE_CORNER corner // значение свойства  
)
```

### Параметры

*corner*  
[in] Новое значение свойства "Угол графика для привязки".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить свойство.

## X\_Size (метод Get)

Получает значение свойства "Размер по оси X".

```
int X_Size() const
```

### Возвращаемое значение

Значение свойства "Размер по оси X" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## X\_Size (метод Set)

Устанавливает значение свойства "Размер по оси X".

```
bool X_Size(  
    int size // значение свойства  
)
```

### Параметры

*size*  
[in] Новое значение свойства "Размер по оси X".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Y\_Size (метод Get)

Получает значение свойства "Размер по оси Y".

```
int Y_Size() const
```

### Возвращаемое значение

Значение свойства "Размер по оси Y" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Y\_Size (метод Set)

Устанавливает значение свойства "Размер по оси Y".

```
bool Y_Size(  
    int size        // значение свойства  
)
```

### Параметры

*size*  
[in] Новое значение свойства "Размер по оси Y".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Symbol (метод Get)

Получает значение свойства "Символ".

```
string Symbol() const
```

### Возвращаемое значение

Значение свойства "Символ" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается "".

## Symbol (метод Set)

Устанавливает значение свойства "Символ".

```
bool Symbol(  
    string symbol // символ  
)
```

### Параметры

*symbol*  
[in] Новое значение свойства "Символ".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Period (метод Get)

Получает значение свойства "Период".

```
int Period() const
```

### Возвращаемое значение

Значение свойства "Период" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Period (метод Set)

Устанавливает значение свойства "Период".

```
bool Period(  
    int period // период  
)
```

### Параметры

*period*

[in] Новое значение свойства "Период".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Scale (метод Get)

Получает значение свойства "Масштаб".

```
double Scale() const
```

### Возвращаемое значение

Значение свойства "Масштаб" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается EMPTY\_VALUE.

## Scale (метод Set)

Устанавливает значение свойства "Масштаб".

```
bool Scale(  
    double scale      // значение свойства  
)
```

### Параметры

*scale*  
[in] Новое значение свойства "Масштаб".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## DateScale (метод Get)

Получает значение флага "Отображение шкалы времени".

```
bool DateScale() const
```

### Возвращаемое значение

Значение флага "Отображение шкалы времени" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается false.

## DateScale (метод Set)

Устанавливает значение флага "Отображение шкалы времени".

```
bool DateScale(  
    bool scale        // значение флага  
)
```

### Параметры

*scale*

[in] Новое значение флага "Отображение шкалы времени".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## PriceScale (метод Get)

Получает значение флага "Отображение ценовой шкалы".

```
bool 1_PriceScale() const
```

### Возвращаемое значение

Значение флага "Отображение ценовой шкалы" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается false.

## PriceScale (метод Set)

Устанавливает значение флага "Отображение ценовой шкалы".

```
bool 2_PriceScale(  
    bool scale // значение флага  
)
```

### Параметры

*scale*

[in] Новое значение флага "Отображение ценовой шкалы".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## Time

Запрещает изменение координаты времени.

```
bool Time(
    datetime time        // любое значение
)
```

### Параметры

*time*

[in]

### Возвращаемое значение

всегда false.

## Price

Запрещает изменение координаты цены.

```
bool Price(
    double price           // любое значение
)
```

### Параметры

*price*  
[in]

### Возвращаемое значение

всегда false.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectSubChart](#) - OBJ\_CHART).

## CChartObjectBitmap

Класс CChartObjectBitmap является классом для упрощенного доступа к свойствам графического объекта "Рисунок".

### Описание

Класс CChartObjectBitmap обеспечивает доступ к свойствам объекта "Рисунок".

### Декларация

```
class CChartObjectBitmap : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsBmpControls.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectBitmap
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Рисунок"
Свойства	
<a href="#">BmpFile</a>	Получает/устанавливает свойство "Имя BMP-файла"
<a href="#">X_Offset</a>	Получает/устанавливает свойство "Х-координата угла области видимости"
<a href="#">Y_Offset</a>	Получает/устанавливает свойство "Y-координата угла области видимости"
Ввод/вывод	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#),

[Z\\_Order](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Смотри также

[Типы объектов](#), [Свойства объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Рисунок".

```
bool Create(
    long      chart_id,      // идентификатор чарта
    string    name,          // имя объекта
    int       window,        // окно чарта
    datetime  time,          // координата времени
    double    price          // координата цены
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*time*

[in] Координата времени точки привязки.

*price*

[in] Координата цены точки привязки.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## BmpFile (метод Get)

Получает значение свойства "Имя BMP-файла".

```
string BmpFile() const
```

### Возвращаемое значение

Значение свойства "Имя BMP-файла" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается "".

## BmpFile (метод Set)

Устанавливает значение свойства "Имя BMP-файла".

```
bool BmpFile(  
    string name // значение свойства  
)
```

### Параметры

*name*  
[in] Новое значение свойства "Имя BMP-файла".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## X\_Offset (метод Get)

Получает значение свойства "Х-координата области видимости" (левого верхнего угла) графического объекта [CChartObjectBitmap](#).

```
int X_Offset() const
```

### Возвращаемое значение

Значение свойства "Х-координата области видимости" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## X\_Offset (метод Set)

Устанавливает значение свойства "Х-координата угла области видимости" (левого верхнего угла) графического объекта [CChartObjectBitmap](#). Значение задается в пикселях относительного верхнего левого угла исходного изображения.

```
bool X_Offset(  
    int X // значение свойства  
)
```

### Параметры

*X*

[in] Новое значение свойства "Х-координата угла области видимости".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Y\_Offset (метод Get)

Получает значение свойства "Y-координата области видимости" (левого верхнего угла) графического объекта [CChartObjectBitmap](#).

```
int Y_Offset() const
```

### Возвращаемое значение

Значение свойства "Y-координата области видимости" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Y\_Offset (метод Set)

Устанавливает значение свойства "Y-координата угла области видимости" (левого верхнего угла) графического объекта [CChartObjectBitmap](#). Значение задается в пикселях относительного верхнего левого угла исходного изображения.

```
bool Y_Offset(  
    int Y // значение свойства  
)
```

### Параметры

*Y*

[in] Новое значение свойства "Y-координата угла области видимости".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectBitmap](#) - OBJ\_BITMAP).

## CChartObjectBmpLabel

Класс CChartObjectBmpLabel является классом для упрощенного доступа к свойствам графического объекта "Графическая метка".

### Описание

Класс CChartObjectBmpLabel обеспечивает доступ к свойствам объекта "Графическая метка".

### Декларация

```
class CChartObjectBmpLabel : public CChartObject
```

### Заголовок

```
#include <ChartObjects\ChartObjectsBmpControls.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectBmpLabel
```

### Методы класса по группам

Создание	
<a href="#"><u>Create</u></a>	Создает графический объект "Графическая метка"
<b>Свойства</b>	
<a href="#"><u>X_Distance</u></a>	Получает/устанавливает свойство "Координата X"
<a href="#"><u>Y_Distance</u></a>	Получает/устанавливает свойство "Координата Y"
<a href="#"><u>X_Offset</u></a>	Получает/устанавливает свойство "X-координата угла области видимости"
<a href="#"><u>Y_Offset</u></a>	Получает/устанавливает свойство "Y-координата угла области видимости"
<a href="#"><u>Corner</u></a>	Получает/устанавливает свойство "Угол графика для привязки"
<a href="#"><u>X_Size</u></a>	Получает свойство "Размер по X"
<a href="#"><u>Y_Size</u></a>	Получает свойство "Размер по Y"
<a href="#"><u>BmpFileOn</u></a>	Получает/устанавливает свойство "Имя BMP-файла" для состояние On (Нажата)

<a href="#"><u>BmpFileOff</u></a>	Получает/устанавливает свойство "Имя BMP-файла" для состояние Off (Отжата)
<a href="#"><u>State</u></a>	Получает/устанавливает состояние кнопки (Нажата/Отжата)
<a href="#"><u>Time</u></a>	"Заглушка" изменения координаты времени
<a href="#"><u>Price</u></a>	"Заглушка" изменения координаты цены
<b>Ввод/вывод</b>	
<a href="#"><u>virtual Save</u></a>	Виртуальный метод записи в файл
<a href="#"><u>virtual Load</u></a>	Виртуальный метод чтения из файла
<a href="#"><u>virtual Type</u></a>	Виртуальный метод идентификации

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CChartObject**

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [SetString](#), [SetString](#), [ShiftObject](#), [ShiftPoint](#)

**Смотри также**[Типы объектов](#), [Свойства объектов](#), [Угол привязки](#), [Графические объекты](#)

## Create

Создает графический объект "Графическая метка".

```
bool Create(
    long    chart_id,      // идентификатор чарта
    string  name,         // имя объекта
    int     window,        // окно чарта
    int     X,             // координата X
    int     Y              // координата Y
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта (0 - текущий чарт).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*X*

[in] Дистанция по оси X.

*Y*

[in] Дистанция по оси Y.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## X\_Distance (метод Get)

Получает значение свойства "Дистанция по оси X".

```
int X_Distance() const
```

### Возвращаемое значение

Значение свойства "Дистанция по оси X" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## X\_Distance (метод Set)

Устанавливает значение свойства "Дистанция по оси X".

```
bool X_Distance(  
    int X // значение свойства  
)
```

### Параметры

X

[in] Новое значение свойства "Дистанция по оси X".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Y\_Distance (метод Get)

Получает значение свойства "Дистанция по оси Y".

```
int Y_Distance() const
```

### Возвращаемое значение

Значение свойства "Дистанция по оси Y" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Y\_Distance (метод Set)

Устанавливает значение свойства "Дистанция по оси Y".

```
bool Y_Distance(  
    int Y // значение свойства  
)
```

### Параметры

*Y*

[in] Новое значение свойства "Дистанция по оси Y".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## X\_Offset (метод Get)

Получает значение свойства "Х-координата области видимости" (левого верхнего угла) графического объекта [CChartObjectBmpLabel](#).

```
int X_Offset() const
```

### Возвращаемое значение

Значение свойства "Х-координата области видимости" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## X\_Offset (метод Set)

Устанавливает значение свойства "Х-координата угла области видимости" (левого верхнего угла) графического объекта [CChartObjectBmpLabel](#). Значение задается в пикселях относительного верхнего левого угла исходного изображения.

```
bool X_Offset(  
    int X // значение свойства  
)
```

### Параметры

*X*

[in] Новое значение свойства "Х-координата угла области видимости".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Y\_Offset (метод Get)

Получает значение свойства "Y-координата области видимости" (левого верхнего угла) графического объекта [CChartObjectBmpLabel](#).

```
int Y_Offset() const
```

### Возвращаемое значение

Значение свойства "Y-координата области видимости" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Y\_Offset (метод Set)

Устанавливает значение свойства "Y-координата угла области видимости" (левого верхнего угла) графического объекта [CChartObjectBmpLabel](#). Значение задается в пикселях относительного верхнего левого угла исходного изображения.

```
bool Y_Offset(  
    int Y // значение свойства  
)
```

### Параметры

*Y*

[in] Новое значение свойства "Y-координата угла области видимости".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Corner (метод Get)

Получает значение свойства "Угол графика для привязки".

```
ENUM_BASE_CORNER Corner() const
```

### Возвращаемое значение

Значение свойства "Угол графика для привязки" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается `WRONG_VALUE`.

## Corner (метод Set)

Устанавливает значение свойства "Угол графика для привязки".

```
bool Corner(  
    ENUM_BASE_CORNER corner // значение свойства  
)
```

### Параметры

*corner*  
[in] Новое значение свойства "Угол графика для привязки".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить свойство.

## X\_Size

Получает значение свойства "Размер по оси X".

```
int X_Size() const
```

### Возвращаемое значение

Значение свойства "Размер по оси X" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## Y\_Size

Получает значение свойства "Размер по оси Y".

```
int Y_Size() const
```

### Возвращаемое значение

Значение свойства "Размер по оси Y" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## BmpFileOn (метод Get)

Получает значение свойства "Имя BMP-файла ON".

```
string BmpFileOn() const
```

### Возвращаемое значение

Значение свойства "Имя BMP-файла ON" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается "".

## BmpFileOn (метод Set)

Устанавливает значение свойства "Имя BMP-файла ON".

```
bool BmpFileOn(  
    string name // имя файла  
)
```

### Параметры

*name*  
[in] Новое значение свойства "Имя BMP-файла ON".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## BmpFileOff (метод Get)

Получает значение свойства "Имя BMP-файла OFF".

```
string BmpFileOff() const
```

### Возвращаемое значение

Значение свойства "Имя BMP-файла OFF" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается "".

## BmpFileOff (метод Set)

Устанавливает значение свойства "Имя BMP-файла OFF".

```
bool BmpFileOff(  
    string name // имя файла  
)
```

### Параметры

*name*  
[in] Новое значение свойства "Имя BMP-файла OFF".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## State (метод Get)

Получает значение свойства "Состояние".

```
bool State() const
```

### Возвращаемое значение

Значение свойства "Состояние" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается false.

## State (метод Set)

Устанавливает значение свойства "Состояние".

```
bool State(  
    bool state // значение свойства  
)
```

### Параметры

state

[in] Новое значение свойства "Состояние".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Time

Запрещает изменение координаты времени.

```
bool Time(
    datetime time          // любое значение
)
```

### Параметры

*time*

[in] Любое значение типа datetime.

### Возвращаемое значение

всегда false.

## Price

Запрещает изменение координаты цены.

```
bool Price(
    double price           // любое значение
)
```

### Параметры

*price*  
[in] Любое значение типа double.

### Возвращаемое значение

всегда false.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectBmpLabel](#) - OBJ\_BITMAP\_LABEL).

## CChartObjectRectLabel

Класс CChartObjectRectLabel является классом для упрощенного доступа к свойствам графического объекта "Прямоугольная метка".

### Описание

Класс CChartObjectRectLabel обеспечивает доступ к свойствам объекта "Прямоугольная метка".

### Декларация

```
class CChartObjectRectLabel : public CChartObjectLabel
```

### Заголовок

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

### Иерархия наследования

```
CObject
  CChartObject
    CChartObjectText
      CChartObjectLabel
        CChartObjectRectLabel
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает графический объект "Прямоугольная метка"
Свойства	
<a href="#">X_Size</a>	Устанавливает свойство "Размер по X"
<a href="#">Y_Size</a>	Устанавливает свойство "Размер по Y"
<a href="#">BackColor</a>	Получает/устанавливает свойство "Цвет фона"
<a href="#">Angle</a>	Запрещает изменение свойства "Угол наклона"
<a href="#">BorderType</a>	Получает/устанавливает свойство "Тип рамки"
Ввод/вывод	
<a href="#">virtual Save</a>	Виртуальный метод записи в файл
<a href="#">virtual Load</a>	Виртуальный метод чтения из файла
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CChartObject

[ChartId](#), [Window](#), [Name](#), [Name](#), [NumPoints](#), [Attach](#), [SetPoint](#), [Delete](#), [Detach](#), [Time](#), [Time](#), [Price](#), [Price](#), [Color](#), [Color](#), [Style](#), [Style](#), [Width](#), [Width](#), [Background](#), [Background](#), [Fill](#), [Fill](#), [Z\\_Order](#), [Z\\_Order](#), [Selected](#), [Selected](#), [Selectable](#), [Selectable](#), [Description](#), [Description](#), [Tooltip](#), [Tooltip](#), [Timeframes](#), [Timeframes](#), [CreateTime](#), [LevelsCount](#), [LevelsCount](#), [LevelColor](#), [LevelColor](#), [LevelStyle](#), [LevelStyle](#), [LevelWidth](#), [LevelWidth](#), [LevelValue](#), [LevelValue](#), [LevelDescription](#), [LevelDescription](#), [GetInteger](#), [GetInteger](#), [SetInteger](#), [SetInteger](#), [GetDouble](#), [GetDouble](#), [SetDouble](#), [SetDouble](#), [GetString](#), [GetString](#), [GetString](#), [ShiftObject](#), [ShiftPoint](#)

#### Методы унаследованные от CChartObjectText

[Angle](#), [Angle](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [Anchor](#), [Anchor](#), [Create](#)

#### Методы унаследованные от CChartObjectLabel

[X\\_Distance](#), [X\\_Distance](#), [Y\\_Distance](#), [Y\\_Distance](#), [X\\_Size](#), [Y\\_Size](#), [Corner](#), [Corner](#), [Time](#), [Price](#), [Create](#)

#### Смотри также

[Типы объектов](#), [Свойства объектов](#), [Графические объекты](#)

## Create

Создает графический объект "Прямоугольная метка".

```
bool Create(
    long    chart_id,      // идентификатор графика
    string  name,         // имя объекта
    int     window,        // окно графика
    int     X,             // координата X
    int     Y,             // координата Y
    int     sizeX,         // размер X
    int     sizeY          // размер Y
)
```

### Параметры

*chart\_id*

[in] Идентификатор графика (0 - текущий график).

*name*

[in] Уникальное имя создаваемого объекта.

*window*

[in] Номер окна чарта (0 - основное окно).

*X*

[in] Дистанция по оси X.

*Y*

[in] Дистанция по оси Y.

*sizeX*

[in] Размер по оси X.

*sizeY*

[in] Размер по оси Y.

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## X\_Size

Устанавливает значение свойства "Размер по оси X".

```
bool X_Size(  
    int size // значение свойства  
)
```

### Параметры

*size*

[in] Новое значение свойства "Размер по оси X".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

### Примечание

Получение значений свойств "Размер по оси X" и "Размер по оси Y" осуществляется при помощи методов [X\\_Size](#) и [Y\\_Size](#) родительского класса [CChartObjectLabel](#).

## Y\_Size

Устанавливает значение свойства "Размер по оси Y".

```
bool Y_Size(  
    int size // значение свойства  
)
```

### Параметры

*size*

[in] Новое значение свойства "Размер по оси Y".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

### Примечание

Получение значений свойств "Размер по оси X" и "Размер по оси Y" осуществляется при помощи методов [X\\_Size](#) и [Y\\_Size](#) родительского класса [CChartObjectLabel](#).

## BackColor

Получает значение свойства "Цвет фона".

```
color BackColor() const
```

### Возвращаемое значение

Значение свойства "Цвет фона" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## BackColor

Устанавливает значение свойства "Цвет фона".

```
bool BackColor(  
    color new_color // значение свойства  
)
```

### Параметры

*new\_color*

[in] Новое значение свойства "Цвет фона".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Angle

Запрещает изменение свойства "Угол наклона".

```
bool Angle(
    double angle           // любое значение
)
```

### Параметры

*angle*  
[in] Любое значение типа [double](#).

### Возвращаемое значение

Всегда false.

## BorderType

Получает значение свойства "Тип рамки".

```
int BorderType() const
```

### Возвращаемое значение

Значение свойства "Тип рамки" объекта, привязанного к экземпляру класса. Если нет привязанного объекта, возвращается 0.

## BorderType

Устанавливает значение свойства "Тип рамки".

```
bool BorderType(  
    int type // значение свойства  
)
```

### Параметры

*type*

[in] Новое значение свойства "Тип рамки".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## Save

Сохраняет параметры объекта в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл бинарного файла, ранее открытого при помощи функции [FileOpen](#).

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Типе

Получает идентификатор типа графического объекта.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа объекта (для [CChartObjectRectLabel](#) - OBJ\_RECTANGLE\_LABEL).

## Пользовательская графика

В этом разделе представлены инструменты для работы с пользовательской графикой.

Их использование существенно упрощает построение пользовательских графиков, рисунков и визуализацию данных.

Отдельно разработаны классы для создания графических объектов и примитивов, для отрисовки различных видов диаграмм и кривых. Реализованы различные возможности отображения объектов: изменение стиля и цвета линий, заливка, работа с сериями данных на графике и т.д.

Класс	Описание
<a href="#">CCanvas</a>	Класс для упрощенного создания пользовательских рисунков
<a href="#">CChartCanvas</a>	Базовый класс для реализации классов, предназначенных для отрисовки графиков и их элементов
<a href="#">CHistogramChart</a>	Класс для построения столбчатых гистограмм
<a href="#">CLineChart</a>	Класс для отрисовки кривых
<a href="#">CPieChart</a>	Класс для построения круговых диаграмм

## CCanvas

Класс CCanvas является классом для упрощенного создания пользовательских рисунков.

### Описание

Класс CCanvas обеспечивает создание графического ресурса (с привязкой к объекту чарта или без) и рисование графических примитивов.

### Декларация

```
class CCanvas
```

### Заголовок

```
#include <Canvas\Canvas.mqh>
```

### Иерархия наследования

CCanvas

#### Прямые потомки

[CChartCanvas](#), [CFlameCanvas](#)

### Методы класса по группам

Создание	
<a href="#">Attach</a>	Привязывает объект OBJ_BITMAP_LABEL к экземпляру класса CCanvas
<a href="#">Create</a>	Создает графический ресурс без привязки к объекту чарта
<a href="#">CreateBitmap</a>	Создает графический ресурс, привязанный к объекту чарта
<a href="#">CreateBitmapLabel</a>	Создает графический ресурс, привязанный к объекту чарта
<a href="#">Destroy</a>	Уничтожает графический ресурс
Свойства	
<a href="#">ChartObjectName</a>	Получает имя привязанного объекта чарта
<a href="#">ResourceName</a>	Получает имя графического ресурса
<a href="#">Width</a>	Получает ширину графического ресурса
<a href="#">Height</a>	Получает высоту графического ресурса
<a href="#">LineStyleSet</a>	Устанавливает стиль линии
Обновление объекта на экране	

<a href="#">Update</a>	Отображает изменения на экран
<a href="#">Resize</a>	Изменяет размеры графического ресурса
<b>Очистка/заполнение цветом</b>	
<a href="#">Erase</a>	Очищает или заполняет указанным цветом
<b>Доступ к данным</b>	
<a href="#">PixelGet</a>	Получает цвет точки с указанными координатами
<a href="#">PixelSet</a>	Устанавливает цвет точки с указанными координатами
<b>Рисование примитивов</b>	
<a href="#">LineVertical</a>	Рисует вертикальную линию
<a href="#">LineHorizontal</a>	Рисует горизонтальную линию
<a href="#">Line</a>	Рисует произвольную линию
<a href="#">Polyline</a>	Рисует ломаную линию
<a href="#">Polygon</a>	Рисует многоугольник
<a href="#">Rectangle</a>	Рисует прямоугольник
<a href="#">Circle</a>	Рисует окружность
<a href="#">Triangle</a>	Рисует треугольник
<a href="#">Ellipse</a>	Рисует эллипс
<a href="#">Arc</a>	Рисует дугу эллипса
<a href="#">Pie</a>	Рисует сектор эллипса
<b>Рисование закрашенных примитивов</b>	
<a href="#">FillRectangle</a>	Рисует закрашенный прямоугольник
<a href="#">FillCircle</a>	Рисует закрашенный круг
<a href="#">FillTriangle</a>	Рисует закрашенный треугольник
<a href="#">FillPolygon</a>	Рисует закрашенный многоугольник
<a href="#">FillEllipse</a>	Рисует закрашенный эллипс
<a href="#">Fill</a>	Закрашивает область
<b>Рисование примитивов с использованием сглаживания</b>	
<a href="#">PixelSetAA</a>	Рисует точку
<a href="#">LineAA</a>	Рисует линию

<a href="#"><u>PolylineAA</u></a>	Рисует ломаную линию
<a href="#"><u>PolygonAA</u></a>	Рисует многоугольник
<a href="#"><u>TriangleAA</u></a>	Рисует треугольник
<a href="#"><u>CircleAA</u></a>	Рисует окружность
<a href="#"><u>EllipseAA</u></a>	Рисует эллипс
<a href="#"><u>LineWu</u></a>	Рисует линию
<a href="#"><u>PolylineWu</u></a>	Рисует ломаную линию
<a href="#"><u>PolygonWu</u></a>	Рисует многоугольник
<a href="#"><u>TriangleWu</u></a>	Рисует треугольник
<a href="#"><u>CircleWu</u></a>	Рисует окружность
<a href="#"><u>EllipseWu</u></a>	Рисует эллипс
<a href="#"><u>LineThick</u></a>	Рисует отрезок произвольной линии заданной толщины с использованием алгоритма сглаживания
<a href="#"><u>LineThickVertical</u></a>	Рисует вертикальный отрезок произвольной линии заданной толщины с использованием алгоритма сглаживания
<a href="#"><u>LineThickHorizontal</u></a>	Рисует горизонтальный отрезок произвольной линии заданной толщины с использованием алгоритма сглаживания
<a href="#"><u>PolygonSmooth</u></a>	Рисует многоугольник заданной толщины с использованием двух алгоритмов сглаживания
<a href="#"><u>PolygonThick</u></a>	Рисует многоугольник заданной толщины с использованием алгоритма сглаживания
<a href="#"><u>PolylineSmooth</u></a>	Рисует ломаную линию заданной толщины с использованием двух алгоритмов сглаживания
<a href="#"><u>PolylineThick</u></a>	Рисует ломаную линию заданной толщины с использованием алгоритма сглаживания
<b>Текст</b>	
<a href="#"><u>FontSet</u></a>	Устанавливает параметры шрифта
<a href="#"><u>FontNameSet</u></a>	Устанавливает имя шрифта
<a href="#"><u>FontSizeSet</u></a>	Устанавливает размер шрифта
<a href="#"><u>FontFlagsSet</u></a>	Устанавливает флаги шрифта
<a href="#"><u>FontAngleSet</u></a>	Устанавливает угол наклона шрифта

<a href="#">FontGet</a>	Получает параметры шрифта
<a href="#">FontNameGet</a>	Получает имя шрифта
<a href="#">FontSizeGet</a>	Получает размер шрифта
<a href="#">FontFlagsGet</a>	Получает флаги шрифта
<a href="#">FontAngleGet</a>	Получает угол наклона шрифта
<a href="#">TextOut</a>	Выводит текст
<a href="#">TextWidth</a>	Получает ширину текста
<a href="#">TextHeight</a>	Получает высоту текста
<a href="#">TextSize</a>	Получает размеры текста
<b>Прозрачность</b>	
<a href="#">TransparentLevelSet</a>	Устанавливает уровень прозрачности
<b>Ввод/вывод</b>	
<a href="#">LoadFromFile</a>	Читает рисунок из BMP-файла

## Attach

Получает из объекта [OBJ\\_BITMAP\\_LABEL](#) графический ресурс и привязывает его к экземпляру класса [CCanvas](#).

```
bool Attach(
    const long      chart_id,                                // идентификатор чарта
    const string     objname,                                // имя объекта
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA      // способ обработки цвета
)
```

Создает графический [ресурс](#) для объекта [OBJ\\_BITMAP\\_LABEL](#) и привязывает его к экземпляру класса [CCanvas](#).

```
bool Attach(
    const long      chart_id,                                // идентификатор чарта
    const string     objname,                                // имя объекта
    const int        width,                                 // ширина картинки в пикселях
    const int        height,                               // высота картинки в пикселях
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA      // способ обработки цвета
)
```

### Параметры

*chart\_id*

[out] Идентификатор чарта.

*objname*

[in] Наименование (имя) графического объекта.

*width*

[in] Ширина картинки в ресурсе.

*height*

[in] Высота картинки в ресурсе.

*clrfmt=COLOR\_FORMAT\_XRGB\_NOALPHA*

[in] Способ обработки альфа-канала. По умолчанию альфа-канал игнорируется.

### Возвращаемое значение

true - в случае удачи, false - если не удалось привязать объект.

## Arc

Рисует дугу эллипса, вписанного в прямоугольник с углами в (x1,y1) и (x2,y2). Границы дуги отсекаются линиями из центра эллипса, идущими к двум точкам с координатами (x3,y3) и (x4,y4).

```
void Arc(
    int      x1,          // координата X верхнего левого угла прямоугольника
    int      y1,          // координата Y верхнего левого угла прямоугольника
    int      x2,          // координата X нижнего правого угла прямоугольника
    int      y2,          // координата Y нижнего правого угла прямоугольника
    int      x3,          // координата X первой точки для нахождения границы дуги
    int      y3,          // координата Y первой точки для нахождения границы дуги
    int      x4,          // координата X второй точки для нахождения границы дуги
    int      y4,          // координата Y второй точки для нахождения границы дуги
    const uint clr        // цвет
);
```

### Параметры

x1

[in] Координата X левого верхнего угла, определяющего прямоугольник.

y1

[in] Координата Y левого верхнего угла, определяющего прямоугольник.

x2

[in] Координата X правого нижнего угла, определяющего прямоугольник.

y2

[in] Координата Y правого нижнего угла, определяющего прямоугольник.

x3

[in] Координата X первой точки, к которой проведена линия из центра прямоугольника для получения границы дуги.

y3

[in] Координата Y первой точки, к которой проведена линия из центра прямоугольника для получения границы дуги.

x4

[in] Координата X второй точки, к которой проведена линия из центра прямоугольника для получения границы дуги.

y4

[in] Координата Y второй точки, к которой проведена линия из центра прямоугольника для получения границы дуги.

clr

[in] Цвет в формате ARGB. Для преобразования цвета в формат ARGB используйте функцию [ColorToARGB\(\)](#).

Рисует дугу эллипса с центром в точке (x,y), вписанного в прямоугольник с радиусами rx и ry. Границы дуги отсекаются лучами из центра эллипса, заданными углами fi3 и fi4.

```
void Arc(
    int      x,          // координата X центра эллипса
    int      y,          // координата Y центра эллипса
    int      rx,         // радиус эллипса по координате X
    int      ry,         // радиус эллипса по координате Y
    int      fi3,        // угол луча из центра эллипса, задающий первую границу дуги
    int      fi4,        // угол луча из центра эллипса, задающий вторую границу дуги
    const uint clr       // цвет
);
```

Рисует дугу эллипса с центром в точке (x,y), вписанного в прямоугольник с радиусами rx и ry, и возвращает координаты границ дуги. Границы дуги отсекаются лучами из центра эллипса, заданными углами fi3 и fi4.

```
void Arc(
    int      x,          // координата X центра эллипса
    int      y,          // координата Y центра эллипса
    int      rx,         // радиус эллипса по координате X
    int      ry,         // радиус эллипса по координате Y
    int      fi3,        // угол луча из центра эллипса, задающий первую границу дуги
    int      fi4,        // угол луча из центра эллипса, задающий вторую границу дуги
    int&    x3,         // координата X первой границы дуги
    int&    y3,         // координата Y первой границы дуги
    int&    x4,         // координата X второй границы дуги
    int&    y4,         // координата Y второй границы дуги
    const uint clr       // цвет
);
```

## Параметры

*x*

[in] Координата X центра эллипса.

*y*

[in] Координата Y центра эллипса.

*rx*

[in] Радиус эллипса по координате X, в пикселях.

*ry*

[in] Радиус эллипса по координате Y, в пикселях.

*fi3*

[in] Угол в радианах, задающий первую границу дуги

*fi4*

[in] Угол в радианах, задающий вторую границу дуги

*x3*

[out] Переменная для получения координаты X первой границы дуги.

*x3*

[out] Переменная для получения координаты Y первой границы дуги.

*x4*

[out] Переменная для получения координаты X второй границы дуги.

*y4*

[out] Переменная для получения координаты Y второй границы дуги.

*clr*

[in] Цвет в формате ARGB. Для преобразования цвета в формат ARGB используйте функцию [ColorToARGB\(\)](#).

Примеры вызова методов класса:

```
#include <Canvas\Canvas.mqh>
CCanvas canvas;
//-----
//| Script program start function
//+-----+
void OnStart()
{
    int      Width=600;
    int      Height=400;
//--- create canvas
    if(!canvas.CreateBitmapLabel(0,0,"CirclesCanvas",30,30,Width,Height))
    {
        Print("Error creating canvas: ",GetLastError());
    }
//--- clear canvas
    canvas.Erase(clrWhite);
//--- draw rectangle
    canvas.Rectangle(215-190,215-120,215+190,215+120,clrGray);
//--- draw first arc
    canvas.Arc(215,215, 190,120,M_PI_4,2*M_PI-M_PI_4,ColorToARGB(clrRed));
    int x1,y1,x2,y2;
//--- draw second arc
    canvas.Arc(215,215, 190,120,2*M_PI-M_PI_4,2*M_PI+M_PI_4,x1,y1,x2,y2,ColorToARGB(clrBlue));
//--- print coordinates of arc
    PrintFormat("First point of arc at (%G,%G), second point of arc at (%G,%G)",x1,y1,x2,y2);
    canvas.CircleAA(x1,y1,3, ColorToARGB(clrRed));
    canvas.CircleAA(x2,y2,3, ColorToARGB(clrBlue));
//--- show updated canvas
    canvas.Update();
}
```

## Pie

Рисует закрашенный сектор эллипса, вписанного в прямоугольник с углами в (x1,y1) и (x2,y2). Границы сектора отсекаются линиями из центра эллипса, идущими к двум точкам с координатами (x3,y3) и (x4,y4).

```
void Pie(
    int      x1,          // координата X верхнего левого угла прямоугольника
    int      y1,          // координата Y верхнего левого угла прямоугольника
    int      x2,          // координата X нижнего правого угла прямоугольника
    int      y2,          // координата Y нижнего правого угла прямоугольника
    int      x3,          // координата X первой точки для нахождения границы дуги
    int      y3,          // координата Y первой точки для нахождения границы дуги
    int      x4,          // координата X второй точки для нахождения границы дуги
    int      y4,          // координата Y второй точки для нахождения границы дуги
    const uint clr,       // цвет линии
    const uint fill_clr // цвет заливки
);
```

### Параметры

x1

[in] Координата X левого верхнего угла, определяющего прямоугольник.

y1

[in] Координата Y левого верхнего угла, определяющего прямоугольник.

x2

[in] Координата X правого нижнего угла, определяющего прямоугольник.

y2

[in] Координата Y правого нижнего угла, определяющего прямоугольник.

x3

[in] Координата X первой точки, к которой проведена линия из центра прямоугольника для получения границы дуги.

y3

[in] Координата Y первой точки, к которой проведена линия из центра прямоугольника для получения границы дуги.

x4

[in] Координата X второй точки, к которой проведена линия из центра прямоугольника для получения границы дуги.

y4

[in] Координата Y второй точки, к которой проведена линия из центра прямоугольника для получения границы дуги.

clr

[in] Цвет границы сектора в формате ARGB.

fill\_clr

[in] Цвет заливки сектора в формате ARGB. Для преобразования цвета в формат ARGB используйте функцию [ColorToARGB\(\)](#).

Рисует закрашенный сектор эллипса с центром в точке (x,y), вписанного в прямоугольник с радиусами rx и ry. Границы сектора отсекаются лучами из центра эллипса, заданными углами fi3 и fi4.

```
void Pie(
    int      x,          // координата X центра эллипса
    int      y,          // координата Y центра эллипса
    int      rx,         // радиус эллипса по координате X
    int      ry,         // радиус эллипса по координате Y
    int      fi3,        // угол луча из центра эллипса, задающий первую границу дуги
    int      fi4,        // угол луча из центра эллипса, задающий вторую границу дуги
    const uint clr,      // цвет линии
    const uint fill_clr // цвет заливки
);
```

Рисует закрашенный сектор эллипса с центром в точке (x,y), вписанного в прямоугольник с радиусами rx и ry, и возвращает координаты границ дуги. Границы сектора отсекаются лучами из центра эллипса, заданными углами fi3 и fi4.

```
void Pie(
    int      x,          // координата X центра эллипса
    int      y,          // координата Y центра эллипса
    int      rx,         // радиус эллипса по координате X
    int      ry,         // радиус эллипса по координате Y
    int      fi3,        // угол луча из центра эллипса, задающий первую границу дуги
    int      fi4,        // угол луча из центра эллипса, задающий вторую границу дуги
    int&    x3,         // координата X первой границы дуги
    int&    y3,         // координата Y первой границы дуги
    int&    x4,         // координата X второй границы дуги
    int&    y4,         // координата Y второй границы дуги
    const uint clr,      // цвет линии
    const uint fill_clr // цвет заливки
);
```

## Параметры

*x*

[in] Координата X центра эллипса.

*y*

[in] Координата Y центра эллипса.

*rx*

[in] Радиус эллипса по координате X, в пикселях.

*ry*

[in] Радиус эллипса по координате X, в пикселях.

*fi3*

[in] Угол в радианах, задающий первую границу дуги

*fi4*

[in] Угол в радианах, задающий вторую границу дуги

*x3*

[out] Переменная для получения координаты X первой границы дуги.

*y3*

[out] Переменная для получения координаты Y первой границы дуги.

*x4*

[out] Переменная для получения координаты X второй границы дуги.

*y4*

[out] Переменная для получения координаты Y второй границы дуги.

*clr*

[in] Цвет границы сектора в формате ARGB.

*fill\_clr*

[in] Цвет заливки сектора в формате ARGB. Для преобразования цвета в формат ARGB используйте функцию [ColorToARGB\(\)](#).

Примеры вызова методов класса:

```
#include <Canvas\Canvas.mqh>
CCanvas canvas;
//-----
//| Script program start function
//+-----+
void OnStart()
{
    int      Width=600;
    int      Height=400;
//--- create canvas
    if(!canvas.CreateBitmapLabel(0,0,"CirclesCanvas",30,30,Width,Height))
    {
        Print("Error creating canvas: ",GetLastError());
    }
//--- clear canvas
    canvas.Erase(clrWhite);
//--- draw rectangle
    canvas.Rectangle(215-190,215-120,215+190,215+120,clrGray);
//--- draw first pie
    canvas.Pie(215,215, 190,120,M_PI_4,2*M_PI-M_PI_4,ColorToARGB(clrBlue),ColorToARGB(c
//--- draw second pie
    canvas.Pie(215,215, 190,120,2*M_PI-M_PI_4,2*M_PI+M_PI_4,ColorToARGB(clrGreen),Color
//--- show updated canvas
```

```
canvas.Update();
DebugBreak();
}
```

## FillPolygon

Рисует закрашенный многоугольник.

```
void FillPolygon(  
    int&      x,          // массив с координатами X точек многоугольника  
    int&      y,          // массив с координатами Y точек многоугольника  
    const uint clr        // цвет  
) ;
```

### Параметры

*x*

[in] Массив, содержащий координаты X точек многоугольника.

*y*

[in] Массив, содержащий координаты Y точек многоугольника.

*clr*

[in] Цвет в формате ARGB.

## FillEllipse

Рисует закрашенный эллипс, вписанный в прямоугольник с заданными координатами.

```
void FillPolygon(
    int      x1,          // координата X верхнего левого угла прямоугольника
    int      y1,          // координата Y верхнего левого угла прямоугольника
    int      x2,          // координата X нижнего правого угла прямоугольника
    int      y2,          // координата Y нижнего правого угла прямоугольника
    const uint clr        // цвет эллипса
);
```

### Параметры

*x1*

[in] Координата X левого верхнего угла, определяющего прямоугольник.

*y1*

[in] Координата Y левого верхнего угла, определяющего прямоугольник.

*x2*

[in] Координата X правого нижнего угла, определяющего прямоугольник.

*y2*

[in] Координата Y правого нижнего угла, определяющего прямоугольник.

*clr*

[in] Цвет в формате ARGB.

## GetDefaultColor

Возвращает предопределенный цвет по его индексу.

```
static uint GetDefaultColor(  
    const uint i           // индекс  
);
```

### Параметры

*i*

[in] Индекс для получения цвета.

### Возвращаемое значение

Цвет.

## ChartObjectName

Получает имя привязанного объекта чарта.

```
string ChartObjectName();
```

### Возвращаемое значение

имя привязанного объекта чарта

## Circle

Рисует окружность

```
void Circle(  
    int      x,           // координата X  
    int      y,           // координата Y  
    int      r,           // радиус  
    const uint clr        // цвет  
>);
```

### Параметры

*x*

[in] Координата X центра окружности.

*y*

[in] Координата Y центра окружности.

*r*

[in] Радиус окружности.

*clr*

[in] Цвет в формате ARGB.

## CircleAA

Рисует окружность с использованием алгоритма сглаживания

```
void CircleAA(  
    const int      x,           // координата X  
    const int      y,           // координата Y  
    const double   r,           // радиус  
    const uint     clr          // цвет  
) ;
```

### Параметры

*x*

[in] Координата X центра окружности.

*y*

[in] Координата Y центра окружности.

*r*

[in] Радиус окружности.

*clr*

[in] Цвет в формате ARGB.

## CircleWu

Рисует окружность с использованием алгоритма сглаживания By

```
void CircleWu(  
    const int      x,           // координата X  
    const int      y,           // координата Y  
    const double   r,           // радиус  
    const uint     clr          // цвет  
) ;
```

### Параметры

*x*

[in] Координата X центра окружности.

*y*

[in] Координата Y центра окружности.

*r*

[in] Радиус окружности.

*clr*

[in] Цвет в формате ARGB.

## Create

Создает графический ресурс без привязки к объекту чарта.

```
virtual bool Create(
    const string      name,                                // имя
    const int         width,                               // ширина
    const int         height,                             // высота
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA   // формат
);
```

### Параметры

*name*

[in] Основание для имени графического ресурса. Имя ресурса формируется при создании путем добавления псевдослучайной строки.

*width*

[in] Ширина (размер по оси X) в пикселях.

*height*

[in] Высота (размер по оси Y) в пикселях.

*clrfmt=COLOR\_FORMAT\_XRGB\_NOALPHA*

[in] Способ обработки цвета. Более подробно о способах обработки цвета смотрите в описании функции [ResourceCreate\(\)](#).

### Возвращаемое значение

true - в случае удачи, иначе false

## CreateBitmap

Создает графический ресурс, привязанный к объекту чарта.

1. Создает графический ресурс в главном окне текущего чарта.

```
bool CreateBitmap(
    const string      name,           // имя
    const datetime    time,          // время
    const double      price,         // цена
    const int         width,        // ширина
    const int         height,       // высота
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // формат
);
```

2. Создает графический ресурс с использованием идентификатора графика и номера подокна.

```
bool CreateBitmap(
    const long        chart_id,      // идентификатор чарта
    const int         subwin,        // номер подокна
    const string      name,          // имя
    const datetime    time,          // время
    const double      price,        // цена
    const int         width,        // ширина
    const int         height,       // высота
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // формат
);
```

### Параметры

*chart\_id*

[in] Идентификатор чарта для создания объекта.

*subwin*

[in] Номер подокна чарта для создания объекта.

*name*

[in] Имя объекта чарта и основание для имени графического ресурса.

*time*

[in] Координата времени точки привязки объекта на чарте.

*price*

[in] Координата цены точки привязки объекта на чарте.

*width*

[in] Ширина графического ресурса (размер по оси X) в пикселях.

*height*

[in] Высота графического ресурса (размер по оси Y) в пикселях.

*clrfmt=COLOR\_FORMAT\_XRGB\_NOALPHA*

[in] Способ обработки цвета. Более подробно о способах обработки цвета смотрите в описании функции [ResourceCreate\(\)](#).

#### Возвращаемое значение

true - в случае удачи, иначе - false

#### Примечание

Если используется первый вариант функции, то объект создается в главном окне текущего чарта.

Размеры объекта совпадают с размерами графического ресурса.

## CreateBitmapLabel

Создает графический ресурс, привязанный к объекту чарта.

1. Создает графический ресурс в главном окне текущего чарта.

```
bool CreateBitmapLabel(
    const string      name,           // имя
    const int         x,              // координата X
    const int         y,              // координата Y
    const int         width,          // ширина
    const int         height,         // высота
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // формат
);
```

2. Создает графический ресурс с использованием идентификатора графика и номера подокна.

```
bool CreateBitmapLabel(
    const long        chart_id,        // идентификатор чарта
    const int         subwin,          // номер подокна
    const string      name,           // имя
    const int         x,              // координата X
    const int         y,              // координата Y
    const int         width,          // ширина
    const int         height,         // высота
    ENUM_COLOR_FORMAT clrfmt=COLOR_FORMAT_XRGB_NOALPHA // формат
);
```

### Параметры

*chart\_id*

[in] Идентификатор чарта для создания объекта.

*subwin*

[in] Номер подокна чарта для создания объекта.

*name*

[in] Имя объекта чарта и основание для имени графического ресурса.

*x*

[in] Координата X точки привязки объекта на чарте.

*y*

[in] Координата Y точки привязки объекта на чарте.

*width*

[in] Ширина графического ресурса (размер по оси X) в пикселях.

*height*

[in] Высота графического ресурса (размер по оси Y) в пикселях.

*clrfmt=COLOR\_FORMAT\_XRGB\_NOALPHA*

[in] Способ обработки цвета. Более подробно о способах обработки цвета смотрите в описании функции [ResourceCreate\(\)](#).

#### Возвращаемое значение

true - в случае удачи, иначе false

#### Примечание

Если используется первый вариант функции, то объект создается в главном окне текущего чарта.

Размеры объекта совпадают с размерами графического ресурса.

## Destroy

Уничтожает графический ресурс.

```
void Destroy();
```

### Примечание

Если графический ресурс создавался с привязкой к объекту чарта, то объект чарта удаляется.

## Ellipse

Рисует эллипс по двум точкам.

```
void Ellipse(
    int      x1,        // координата X
    int      y1,        // координата Y
    int      x2,        // координата X
    int      y2,        // координата Y
    const uint clr       // цвет
);
```

### Параметры

*x1*

[in] Координата X первой точки, определяющей эллипс.

*y1*

[in] Координата Y первой точки, определяющей эллипс.

*x2*

[in] Координата X второй точки, определяющей эллипс.

*y2*

[in] Координата Y второй точки, определяющей эллипс.

*clr*

[in] Цвет в формате ARGB.

## EllipseAA

Рисует эллипс по двум точкам с использованием алгоритма сглаживания.

```
void EllipseAA(
    int      x1,        // координата X
    int      y1,        // координата Y
    int      x2,        // координата X
    int      y2,        // координата Y
    const uint clr       // цвет
);
```

### Параметры

*x1*

[in] Координата X первой точки, определяющей эллипс.

*y1*

[in] Координата Y первой точки, определяющей эллипс.

*x2*

[in] Координата X второй точки, определяющей эллипс.

*y2*

[in] Координата Y второй точки, определяющей эллипс.

*clr*

[in] Цвет в формате ARGB.

## EllipseWu

Рисует эллипс по двум точкам с использованием алгоритма сглаживания By.

```
void Ellipse(
    int      x1,        // координата X
    int      y1,        // координата Y
    int      x2,        // координата X
    int      y2,        // координата Y
    const uint clr       // цвет
);
```

### Параметры

*x1*

[in] Координата X первой точки, определяющей эллипс.

*y1*

[in] Координата Y первой точки, определяющей эллипс.

*x2*

[in] Координата X второй точки, определяющей эллипс.

*y2*

[in] Координата Y второй точки, определяющей эллипс.

*clr*

[in] Цвет в формате ARGB.

## Erase

Очищает или заполняет указанным цветом.

```
void Erase(
    const uint clr=0           // цвет
);
```

### Параметры

*clr=0*

[in] Цвет в формате ARGB.

## Fill

Закрашивает область.

```
void Fill(
    int      x,           // координата X
    int      y,           // координата Y
    const uint clr        // цвет
);
```

### Параметры

*x*

[in] Координата X точки начала закрашивания.

*y*

[in] Координата Y точки начала закрашивания.

*clr*

[in] Цвет в формате ARGB.

## FillCircle

Рисует закрашенный круг.

```
void FillCircle(
    int      x,           // координата X
    int      y,           // координата Y
    int      r,           // радиус
    const uint clr        // цвет
);
```

### Параметры

*x*

[in] Координата X центра круга.

*y*

[in] Координата Y центра круга.

*r*

[in] Радиус круга.

*clr*

[in] Цвет в формате ARGB.

## FillRectangle

Рисует закрашенный прямоугольник.

```
void FillRectangle(
    int      x1,        // координата X
    int      y1,        // координата Y
    int      x2,        // координата X
    int      y2,        // координата Y
    const uint clr       // цвет
);
```

### Параметры

*x1*

[in] Координата X первой точки, определяющей прямоугольник.

*y1*

[in] Координата Y первой точки, определяющей прямоугольник.

*x2*

[in] Координата X второй точки, определяющей прямоугольник.

*y2*

[in] Координата Y второй точки, определяющей прямоугольник.

*clr*

[in] Цвет в формате ARGB.

## FillTriangle

Рисует закрашенный треугольник.

```
void FillTriangle(
    int      x1,        // координата X
    int      y1,        // координата Y
    int      x2,        // координата X
    int      y2,        // координата Y
    int      x3,        // координата X
    int      y3,        // координата Y
    const uint clr       // цвет
);
```

### Параметры

*x1*

[in] Координата X первой вершины треугольника.

*y1*

[in] Координата Y первой вершины треугольника.

*x2*

[in] Координата X второй вершины треугольника.

*y2*

[in] Координата Y второй вершины треугольника.

*x3*

[in] Координата X третьей вершины треугольника.

*y3*

[in] Координата Y третьей вершины треугольника.

*clr*

[in] Цвет в формате ARGB.

## FontAngleGet

Получает угол наклона шрифта.

```
uint FontAngleGet();
```

### Возвращаемое значение

угол наклона шрифта

## FontAngleSet

Устанавливает угол наклона шрифта.

```
bool FontAngleSet(  
    uint angle // угол  
) ;
```

### Параметры

*angle*

[in] Угол наклона шрифта в десятых долях градуса.

### Возвращаемое значение

true - в случае удачи, иначе - false

## FontFlagsGet

Получает флаги шрифта.

```
uint FontFlagsGet();
```

### Возвращаемое значение

флаги шрифта

## FontFlagsSet

Устанавливает флаги шрифта.

```
bool FontFlagsSet(  
    uint flags // флаги  
) ;
```

### Параметры

*flags*

[in] Флаги создания шрифта. Более подробно о флагах смотрите в описании функции [TextSetFont\(\)](#).

### Возвращаемое значение

true - в случае удачи, иначе - false

## FontGet

Получает параметры текущего шрифта.

```
void FontGet(
    string& name,           // имя
    int& size,             // размер
    uint& flags,            // флаги
    uint& angle             // угол наклона
);
```

### Параметры

*name*

[out] Ссылка на переменную для возврата имени шрифта.

*size*

[out] Ссылка на переменную для возврата размера шрифта.

*flags*

[out] Ссылка на переменную для возврата флагов шрифта.

*angle*

[out] Ссылка на переменную для возврата угла наклона шрифта.

## FontNameGet

Получает имя шрифта.

```
string  FontNameGet();
```

### Возвращаемое значение

имя шрифта

## FontNameSet

Устанавливает имя шрифта.

```
bool FontNameSet(
    string name        // имя
);
```

### Параметры

*name*

[in] Имя шрифта. Например, "Arial".

### Возвращаемое значение

true - в случае удачи, иначе - false

## FontSet

Устанавливает текущий шрифт.

```
bool FontSet(
    const string  name,           // имя
    const int     size,           // размер
    const uint    flags=0,        // флаги
    const uint    angle=0         // угол
);
```

### Параметры

*name*

[in] Имя шрифта. Например "Arial".

*size*

[in] Размер шрифта. Более подробно о специфике задания размера смотрите в описании функции [TextSetFont\(\)](#).

*flags=0*

[in] Флаги создания шрифта. Более подробно о флагах смотрите в описании функции [TextSetFont\(\)](#).

*angle=0*

[in] Угол наклона шрифта в десятых долях градуса.

### Возвращаемое значение

true - в случае удачи, иначе - false

## FontSizeGet

Получает размер шрифта.

```
int  FontSizeGet();
```

**Возвращаемое значение**

размер шрифта

## FontSizeSet

Устанавливает размер шрифта.

```
bool FontSizeSet(  
    int size // размер  
) ;
```

### Параметры

*size*

[in] Размер шрифта. Более подробно о специфике задания размера смотрите в описании функции [TextSetFont\(\)](#).

### Возвращаемое значение

true - в случае удачи, иначе - false

## Height

Получает высоту графического ресурса.

```
int Height();
```

### Возвращаемое значение

высота графического ресурса

## Line

Рисует отрезок произвольной линии.

```
void Line(
    int      x1,      // координата X
    int      y1,      // координата Y
    int      x2,      // координата X
    int      y2,      // координата Y
    const uint clr   // цвет
);
```

### Параметры

*x1*

[in] Координата X первой точки отрезка.

*y1*

[in] Координата Y первой точки отрезка.

*x2*

[in] Координата X второй точки отрезка.

*y2*

[in] Координата Y второй точки отрезка.

*clr*

[in] Цвет в формате ARGB.

## LineAA

Рисует отрезок произвольной линии с использованием алгоритма сглаживания.

```
void LineAA(
    const int    x1,           // координата X
    const int    y1,           // координата Y
    const int    x2,           // координата X
    const int    y2,           // координата Y
    const uint   clr,          // цвет
    const uint   style=UINT_MAX // стиль линии
);
```

### Параметры

*x1*

[in] Координата X первой точки отрезка.

*y1*

[in] Координата Y первой точки отрезка.

*x2*

[in] Координата X второй точки отрезка.

*y2*

[in] Координата Y второй точки отрезка.

*clr*

[in] Цвет в формате ARGB.

*style=UINT\_MAX*

[in] Стиль линии - одно из значений перечисления [ENUM\\_LINE\\_STYLE](#) или пользовательское значение.

## LineWu

Рисует отрезок произвольной линии с использованием алгоритма сглаживания By.

```
void LineWu(
    const int    x1,           // координата X
    const int    y1,           // координата Y
    const int    x2,           // координата X
    const int    y2,           // координата Y
    const uint   clr,          // цвет
    const uint   style=UINT_MAX // стиль линии
);
```

### Параметры

*x1*

[in] Координата X первой точки отрезка.

*y1*

[in] Координата Y первой точки отрезка.

*x2*

[in] Координата X второй точки отрезка.

*y2*

[in] Координата Y второй точки отрезка.

*clr*

[in] Цвет в формате ARGB.

*style=UINT\_MAX*

[in] Стиль линии - одно из значений перечисления [ENUM\\_LINE\\_STYLE](#) или пользовательское значение.

## LineHorizontal

Рисует отрезок горизонтальной линии.

```
void LineHorizontal(
    int      x1,          // координата X
    int      x2,          // координата X
    int      y,           // координата Y
    const uint clr        // цвет
);
```

### Параметры

*x1*

[in] Координата X первой точки отрезка.

*x2*

[in] Координата X второй точки отрезка.

*y*

[in] Координата Y отрезка.

*clr*

[in] Цвет в формате ARGB.

## LineVertical

Рисует отрезок вертикальной линии.

```
void LineVertical(
    int      x,          // координата X
    int      y1,         // координата Y
    int      y2,         // координата Y
    const uint clr        // цвет
);
```

### Параметры

*x*

[in] Координата X отрезка.

*y1*

[in] Координата Y первой точки отрезка.

*y2*

[in] Координата Y второй точки отрезка.

*clr*

[in] Цвет в формате ARGB.

## LineStyleSet

Устанавливает стиль линии.

```
void LineStyleSet(
    const uint style      // стиль
);
```

### Параметры

*style*  
[in] Стиль линии.

### Примечание

Входной параметр может принимать любое из значений перечисления ENUM\_LINE\_STYLE. Кроме того, существует возможность создать пользовательский стиль рисования линии.

## LineThick

Рисует отрезок произвольной линии заданной толщины с использованием алгоритма сглаживания.

```
void LineThick(
    const int      x1,           // координата X первой точки отрезка
    const int      y1,           // координата Y первой точки отрезка
    const int      x2,           // координата X второй точки отрезка
    const int      y2,           // координата Y второй точки отрезка
    const uint     clr,          // цвет
    const int      size,         // толщина линии
    const uint     style,        // стиль линии
    ENUM_LINE_END end_style    // стиль концов линии
)
```

### Параметры

*x1*

[in] Координата X первой точки отрезка.

*y1*

[in] Координата Y первой точки отрезка.

*x2*

[in] Координата X второй точки отрезка.

*y2*

[in] Координата Y второй точки отрезка.

*clr*

[in] Цвет в формате ARGB.

*size*

[in] Толщина линии.

*style*

[in] Стиль линии – одно из значений перечисления ENUM\_LINE\_STYLE или пользовательское значение.

*end\_style*

[in] Стиль концов линии – одно из значений перечисления ENUM\_LINE\_END

### ENUM\_LINE\_END

Идентификатор	Описание
LINE_END_ROUND	Концы линий закругляются.
LINE_END_BUTT	Концы линий обрезаются.
LINE_END_SQUARE	Линия оканчивается закрашенным прямоугольником.

## LineThickVertical

Рисует вертикальный отрезок произвольной линии заданной толщины с использованием алгоритма сглаживания.

```
void LineThickVertical(
    const int      x,           // координата X отрезка
    const int      y1,          // координата Y первой точки отрезка
    const int      y2,          // координата Y второй точки отрезка
    const uint     clr,          // цвет
    const int      size,         // толщина линии
    const uint     style,         // стиль линии
    ENUM_LINE_END end_style    // стиль концов линии
)
```

### Параметры

*x*

[in] Координата X отрезка.

*y1*

[in] Координата Y первой точки отрезка.

*y2*

[in] Координата Y второй точки отрезка.

*clr*

[in] Цвет в формате ARGB.

*size*

[in] Толщина линии.

*style*

[in] Стиль линии – одно из значений перечисления ENUM\_LINE\_STYLE или пользовательское значение.

*end\_style*

[in] Стиль концов линии – одно из значений перечисления [ENUM\\_LINE\\_END](#).

## LineThickHorizontal

Рисует горизонтальный отрезок произвольной линии заданной толщины со сглаживанием.

```
void LineThickHorizontal(
    const int      x1,           // координата X первой точки отрезка
    const int      x2,           // координата X второй точки отрезка
    const int      y,            // координата Y отрезка
    const uint     clr,          // цвет
    const int      size,         // толщина линии
    const uint     style,        // стиль линии
    ENUM_LINE_END end_style    // стиль концов линии
)
```

### Параметры

*x1*

[in] Координата X первой точки отрезка.

*x2*

[in] Координата X второй точки отрезка.

*y*

[in] Координата Y отрезка.

*clr*

[in] Цвет в формате ARGB.

*size*

[in] Толщина линии.

*style*

[in] Стиль линии – одно из значений перечисления `ENUM_LINE_STYLE` или пользовательское значение.

*end\_style*

[in] Стиль концов линии – одно из значений перечисления `ENUM_LINE_END`.

## LoadFromFile

Читает рисунок из BMP-файла.

```
bool LoadFromFile(  
    const string filename // имя файла  
) ;
```

### Параметры

*filename*  
[in] Имя файла (включая расширение "BMP").

### Возвращаемое значение

true - в случае удачи, иначе - false

## PixelGet

Получает цвет точки с указанными координатами.

```
uint PixelGet(  
    const int x,      // координата X  
    const int y       // координата Y  
);
```

### Параметры

*x*

[in] Координата X точки.

*y*

[in] Координата Y точки.

### Возвращаемое значение

цвет точки в формате ARGB.

## PixelSet

Устанавливает цвет точки с указанными координатами.

```
void PixelSet(
    const int    x,           // координата X
    const int    y,           // координата Y
    const uint   clr         // цвет
);
```

### Параметры

*x*

[in] Координата X точки.

*y*

[in] Координата Y точки.

*clr*

[in] Цвет в формате ARGB.

## PixelSetAA

Рисует точку с использованием алгоритма сглаживания.

```
void PixelSetAA(
    const double x,           // координата X
    const double y,           // координата Y
    const uint   clr         // цвет
);
```

### Параметры

*x*

[in] Координата X точки.

*y*

[in] Координата Y точки.

*clr*

[in] Цвет в формате ARGB.

## Polygon

Рисует многоугольник.

```
void Polygon(
    int&      x[],      // массив координат X
    int&      y[],      // массив координат Y
    const uint  clr       // цвет
);
```

### Параметры

*x[]*

[in] Массив координат X точек многоугольника.

*y[]*

[in] Массив координат Y точек многоугольника.

*clr*

[in] Цвет в формате ARGB.

## PolygonAA

Рисует многоугольник с использованием алгоритма сглаживания.

```
void PolygonAA(
    int&      x[],           // массив координат X
    int&      y[],           // массив координат Y
    const uint  clr,          // цвет
    const uint  style=UINT_MAX // стиль линии
);
```

### Параметры

*x*[]

[in] Массив координат X точек многоугольника.

*y*[]

[in] Массив координат Y точек многоугольника.

*clr*

[in] Цвет в формате ARGB.

*style=UINT\_MAX*

[in] Стиль линии - одно из значений перечисления [ENUM\\_LINE\\_STYLE](#) или пользовательское значение.

## PolygonWu

Рисует многоугольник с использованием алгоритма сглаживания By.

```
void PolygonWu(
    int&      x[],           // массив координат X
    int&      y[],           // массив координат Y
    const uint  clr,          // цвет
    const uint  style=UINT_MAX // стиль линии
);
```

### Параметры

*x*[]

[in] Массив координат X точек многоугольника.

*y*[]

[in] Массив координат Y точек многоугольника.

*clr*

[in] Цвет в формате ARGB.

*style=UINT\_MAX*

[in] Стиль линии - одно из значений перечисления [ENUM\\_LINE\\_STYLE](#) или пользовательское значение.

## PolygonThick

Рисует многоугольник заданной толщины с использованием алгоритма сглаживания.

```
void PolygonThick(
    const int&      x[],           // массив координат X точек многоугольника
    const int&      y[],           // массив координат Y точек многоугольника
    const uint       clr,          // цвет
    const int        size,          // толщина линии
    const uint       style,          // стиль линии
    ENUM_LINE_END   end_style     // стиль концов линии
)
```

### Параметры

*x[]*

[in] Массив координат X точек многоугольника.

*y[]*

[in] Массив координат Y точек многоугольника.

*clr*

[in] Цвет в формате ARGB.

*size*

[in] Толщина линии.

*style*

[in] Стиль линии – одно из значений перечисления ENUM\_LINE\_STYLE или пользовательское значение.

*end\_style*

[in] Стиль концов линии – одно из значений перечисления ENUM\_LINE\_END.

## PolygonSmooth

Рисует многоугольник заданной толщины с использованием двух алгоритмов сглаживания последовательно. Сначала на основе кривых Безье сглаживаются отдельные отрезки. Затем для повышения качества отрисовки к построенному из этих отрезков многоугольнику применяется растровый алгоритм сглаживания.

```
void PolygonSmooth(
    int&           x[],          // массив координат X точек многоугольника
    int&           y[],          // массив координат Y точек многоугольника
    const uint      clr,          // цвет
    const int       size,         // толщина линии
    ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
    ENUM_LINE_END   end_style=LINE_END_ROUND, // стиль концов линии
    double          tension=0.5,   // значение параметра сглаживания
    double          step=10       // длина аппроксимирующих линий
)
```

### Параметры

*&x[]*

[in] Массив координат X точек многоугольника.

*&y[]*

[in] Массив координат Y точек многоугольника.

*clr*

[in] Цвет в формате ARGB.

*size*

[in] Толщина линии.

*style=STYLE\_SOLID*

[in] Стиль линии – одно из значений перечисления `ENUM_LINE_STYLE` или пользовательское значение.

*end\_style=LINE\_END\_ROUND*

[in] Стиль концов линии – одно из значений перечисления `ENUM_LINE_END`.

*tension=0.5*

[in] Значение параметра сглаживания.

*step=10*

[in] Длина аппроксимирующих линий.

## Polyline

Рисует ломаную линию.

```
void Polyline(  
    int&      x[],      // массив координат X  
    int&      y[],      // массив координат Y  
    const uint clr       // цвет  
) ;
```

### Параметры

*x[]*

[in] Массив координат X точек ломаной линии.

*y[]*

[in] Массив координат Y точек ломаной линии.

*clr*

[in] Цвет в формате ARGB.

## PolylineSmooth

Рисует ломаную линию заданной толщины с использованием двух алгоритмов сглаживания последовательно. Сначала на основе кривых Безье сглаживаются отдельные отрезки линии. Затем для повышения качества отрисовки к построенной из этих отрезков ломаной линии применяется растровый алгоритм сглаживания.

```
void PolylineSmooth(
    const int&           x[],          // массив координат X точек ломаной линии
    const int&           y[],          // массив координат Y точек ломаной линии
    const uint            clr,          // цвет
    const int             size,          // толщина линии
    ENUM_LINE_STYLE       style=STYLE_SOLID, // стиль линии
    ENUM_LINE_END         end_style=LINE_END_ROUND, // стиль концов линии
    double               tension=0.5,    // значение параметра сглаживания
    double               step=10        // шаг аппроксимации
)
```

### Параметры

*&x[]*

[in] Массив координат X точек ломаной линии.

*&y[]*

[in] Массив координат Y точек ломаной линии.

*clr*

[in] Цвет в формате ARGB.

*size*

[in] Толщина линии.

*style=STYLE\_SOLID*

[in] Стиль линии – одно из значений перечисления ENUM\_LINE\_STYLE или пользовательское значение.

*end\_style=LINE\_END\_ROUND*

[in] Стиль концов линии – одно из значений перечисления ENUM\_LINE\_END.

*tension=0.5*

[in] Значение параметра сглаживания.

*step=10*

[in] Шаг аппроксимации.

## PolylineThick

Рисует ломаную линию заданной толщины с использованием алгоритма сглаживания.

```
void PolylineThick(
    const int      &x[],           // массив координат X точек ломаной линии
    const int      &y[],           // массив координат Y точек ломаной линии
    const uint     clr,            // цвет
    const int      size,           // толщина линии
    const uint     style,          // стиль линии
    ENUM_LINE_END end_style       // стиль концов линии
)
```

### Параметры

*&x[]*

[in] Массив координат X точек ломаной линии.

*&y[]*

[in] Массив координат Y точек ломаной линии.

*clr*

[in] Цвет в формате ARGB.

*size*

[in] Толщина линии.

*style*

[in] Стиль линии — одно из значений перечисления `ENUM_LINE_STYLE` или пользовательское значение.

*end\_style*

[in] Стиль концов линии — одно из значений перечисления `ENUM_LINE_END`.

## PolylineWu

Рисует ломаную линию с использованием алгоритма сглаживания Ву.

```
void PolylineWu(
    int&      x[],           // массив координат X
    int&      y[],           // массив координат Y
    const uint  clr,          // цвет
    const uint  style=UINT_MAX // стиль линии
);
```

### Параметры

*x*[]

[in] Массив координат X точек ломаной линии.

*y*[]

[in] Массив координат Y точек ломаной линии.

*clr*

[in] Цвет в формате ARGB.

*style=UINT\_MAX*

[in] Стиль линии - одно из значений перечисления [ENUM\\_LINE\\_STYLE](#) или пользовательское значение.

## PolylineAA

Рисует ломаную линию с использованием алгоритма сглаживания.

```
void PolylineAA(
    int&      x[],           // массив координат X
    int&      y[],           // массив координат Y
    const uint  clr,          // цвет
    const uint  style=UINT_MAX // стиль линии
);
```

### Параметры

*x*[]

[in] Массив координат X точек ломаной линии.

*y*[]

[in] Массив координат Y точек ломаной линии.

*clr*

[in] Цвет в формате ARGB.

*style=UINT\_MAX*

[in] Стиль линии - одно из значений перечисления [ENUM\\_LINE\\_STYLE](#) или пользовательское значение.

## Rectangle

Рисует прямоугольник по двум точкам.

```
void Rectangle(
    int      x1,      // координата X
    int      y1,      // координата Y
    int      x2,      // координата X
    int      y2,      // координата Y
    const uint clr    // цвет
);
```

### Параметры

*x1*

[in] Координата X первой точки, определяющей прямоугольник.

*y1*

[in] Координата Y первой точки, определяющей прямоугольник.

*x2*

[in] Координата X второй точки, определяющей прямоугольник.

*y2*

[in] Координата Y второй точки, определяющей прямоугольник.

*clr*

[in] Цвет в формате ARGB.

## Resize

Изменяет размеры графического ресурса.

```
bool Resize(  
    const int width,           // ширина  
    const int height           // высота  
);
```

### Параметры

*width*

[in] Новая ширина графического ресурса.

*height*

[in] Новая высота графического ресурса.

### Возвращаемое значение

true - в случае удачи, иначе - false

### Примечание

При изменении размеров предыдущее изображение не сохраняется.

## ResourceName

Получает имя графического ресурса.

```
string  ResourceManager();
```

### Возвращаемое значение

имя графического ресурса

## TextHeight

Получает высоту текста.

```
int TextHeight(
    const string text      // текст
);
```

### Параметры

*text*

[in] Текст для измерения.

### Возвращаемое значение

высота текста в пикселях

### Примечание

Для измерения текста используется текущий шрифт.

## TextOut

Выводит текст.

```
void TextOut(
    int      x,           // координата X
    int      y,           // координата Y
    string   text,        // текст
    const uint clr,       // цвет
    uint     alignment=0  // выравнивание
);
```

### Параметры

*x*

[in] Координата X точки привязки текста.

*y*

[in] Координата Y точки привязки текста.

*text*

[in] Текст для отображения.

*clr*

[in] Цвет в формате ARGB.

*alignment=0*

[in] Способ привязки текста. Более подробно о способах привязки смотрите в описании функции работы с графическими объектами [TextOut\(\)](#).

### Примечание

Для вывода текста используется текущий шрифт.

## TextSize

Получает размеры текста.

```
void TextSize(  
    const string  text,          // текст  
    int&         width,         // ширина  
    int&         height         // высота  
) ;
```

### Параметры

*text*

[in] Текст для измерения.

*width*

[out] Ссылка на переменную для возврата ширины текста.

*height*

[out] Ссылка на переменную для возврата высоты текста.

### Примечание

Для измерения текста используется текущий шрифт.

## TextWidth

Получает ширину текста.

```
int TextWidth(
    const string text      // текст
);
```

### Параметры

*text*

[in] Текст для измерения.

### Возвращаемое значение

ширина текста в пикселях

### Примечание

Для измерения текста используется текущий шрифт.

## TransparentLevelSet

Устанавливает уровень прозрачности.

```
void TransparentLevelSet(
    const uchar value      // значение
);
```

### Параметры

*value*

[in] Новое значение уровня прозрачности.

### Примечание

Значение 0 соответствует полной прозрачности, значение 255 - абсолютной непрозрачности.

Установка уровня прозрачности влияет на все ранее нарисованное. На последующие построения установленный уровень прозрачности не распространяется.

## Triangle

Рисует треугольник.

```
void Triangle(
    int      x1,      // координата X
    int      y1,      // координата Y
    int      x2,      // координата X
    int      y2,      // координата Y
    int      x3,      // координата X
    int      y3,      // координата Y
    const uint clr    // цвет
);
```

### Параметры

*x1*

[in] Координата X первой вершины треугольника.

*y1*

[in] Координата Y первой вершины треугольника.

*x2*

[in] Координата X второй вершины треугольника.

*y2*

[in] Координата Y второй вершины треугольника.

*x3*

[in] Координата X третьей вершины треугольника.

*y3*

[in] Координата Y третьей вершины треугольника.

*clr*

[in] Цвет в формате ARGB.

## TriangleAA

Рисует треугольник с использованием алгоритма сглаживания.

```
void TriangleAA(
    const int    x1,           // координата X
    const int    y1,           // координата Y
    const int    x2,           // координата X
    const int    y2,           // координата Y
    const int    x3,           // координата X
    const int    y3,           // координата Y
    const uint   clr,          // цвет
    const uint   style=UINT_MAX // стиль линии
);
```

### Параметры

*x1*

[in] Координата X первой вершины треугольника.

*y1*

[in] Координата Y первой вершины треугольника.

*x2*

[in] Координата X второй вершины треугольника.

*y2*

[in] Координата Y второй вершины треугольника.

*x3*

[in] Координата X третьей вершины треугольника.

*y3*

[in] Координата Y третьей вершины треугольника.

*clr*

[in] Цвет в формате ARGB.

*style=UINT\_MAX*

[in] Стиль линии - одно из значений перечисления [ENUM\\_LINE\\_STYLE](#) или пользовательское значение.

## TriangleWu

Рисует треугольник с использованием алгоритма сглаживания By.

```
void TriangleWu(
    const int    x1,           // координата X
    const int    y1,           // координата Y
    const int    x2,           // координата X
    const int    y2,           // координата Y
    const int    x3,           // координата X
    const int    y3,           // координата Y
    const uint   clr,          // цвет
    const uint   style=UINT_MAX // стиль линии
);
```

### Параметры

*x1*

[in] Координата X первой вершины треугольника.

*y1*

[in] Координата Y первой вершины треугольника.

*x2*

[in] Координата X второй вершины треугольника.

*y2*

[in] Координата Y второй вершины треугольника.

*x3*

[in] Координата X третьей вершины треугольника.

*y3*

[in] Координата Y третьей вершины треугольника.

*clr*

[in] Цвет в формате ARGB.

*style=UINT\_MAX*

[in] Стиль линии - одно из значений перечисления [ENUM\\_LINE\\_STYLE](#) или пользовательское значение.

## Update

Отображает изменения на экран.

```
void Update(
    const bool redraw=true           // флаг
);
```

### Параметры

*redraw=true*

Флаг необходимости перерисовки чарта.

## Width

Получает ширину графического ресурса.

```
int Width();
```

### Возвращаемое значение

ширина графического ресурса

## CChartCanvas

Базовый класс для реализации классов, с помощью которых происходит отрисовка графиков и их элементов.

### Описание

Данный класс включает в себя методы для работы с основными элементами любого графика: координатными осями и их разметкой, легендой графика, сеткой, фоном и т.д. Здесь настраиваются параметры отображения элементов: видимость/невидимость, цвет текста и т.д.

### Декларация

```
class CChartCanvas : public CCanvas
```

### Заголовок

```
#include <Canvas\Charts\ChartCanvas.mqh>
```

### Иерархия наследования

[CCanvas](#)

CChartCanvas

### Прямые потомки

[CHistogramChart](#), [CLineChart](#), [CPieChart](#)

### Методы класса

Метод	Действие
<a href="#">ColorBackground</a>	Возвращает и устанавливает цвет фона.
<a href="#">ColorBorder</a>	Возвращает и устанавливает цвет границ.
<a href="#">ColorText</a>	Возвращает и устанавливает цвет текста.
<a href="#">ColorGrid</a>	Возвращает и устанавливает цвет сетки.
<a href="#">MaxData</a>	Возвращает и устанавливает максимальное количество допустимых данных (серий).
<a href="#">MaxDescrLen</a>	Возвращает и устанавливает максимальную длину дескрипторов.
<a href="#">ShowFlags</a>	Возвращает и устанавливает флаг видимости элементов графика.
<a href="#">IsShowLegend</a>	Возвращает флаг видимости легенды на графике.
<a href="#">IsShowScaleLeft</a>	Возвращает флаг видимости шкалы значений слева.

<a href="#">IsShowScaleRight</a>	Возвращает флаг видимости шкалы значений справа.
<a href="#">IsShowScaleTop</a>	Возвращает флаг видимости шкалы значений сверху.
<a href="#">IsShowScaleBottom</a>	Возвращает флаг видимости шкалы значений снизу.
<a href="#">IsShowGrid</a>	Возвращает флаг видимости сетки на графике.
<a href="#">IsShowDescriptors</a>	Возвращает флаг видимости дескрипторов на графике.
<a href="#">IsShowPercent</a>	Возвращает флаг видимости процентов на графике.
<a href="#">VScaleMin</a>	Возвращает и устанавливает минимум на вертикальной шкале значений.
<a href="#">VScaleMax</a>	Возвращает и устанавливает максимум на вертикальной шкале значений.
<a href="#">NumGrid</a>	Возвращает и устанавливает количество вертикальных делений шкалы при построении сетки графика.
<a href="#">DataOffset</a>	Возвращает и устанавливает значение смещения данных.
<a href="#">DataTotal</a>	Возвращает общее количество серий данных на графике.
<a href="#">DrawDescriptors</a>	Виртуальный метод для отрисовки дескрипторов.
<a href="#">DrawData</a>	Виртуальный метод для отрисовки серии данных по указанному индексу.
<a href="#">Create</a>	Виртуальный метод, который создаёт графический ресурс.
<a href="#">AllowedShowFlags</a>	Устанавливает набор разрешенных флагов видимости для элементов графика.
<a href="#">ShowLegend</a>	Устанавливает флаг видимости для легенды.
<a href="#">ShowScaleLeft</a>	Устанавливает флаг видимости для левой шкалы.
<a href="#">ShowScaleRight</a>	Устанавливает флаг видимости для правой шкалы.
<a href="#">ShowScaleTop</a>	Устанавливает флаг видимости для верхней шкалы.

<a href="#"><u>ShowScaleBottom</u></a>	Устанавливает флаг видимости для нижней шкалы.
<a href="#"><u>ShowGrid</u></a>	Устанавливает флаг видимости для сетки.
<a href="#"><u>ShowDescriptors</u></a>	Устанавливает флаг видимости для дескрипторов.
<a href="#"><u>ShowValue</u></a>	Устанавливает флаг видимости для значений.
<a href="#"><u>ShowPercent</u></a>	Устанавливает флаг видимости для процентов.
<a href="#"><u>LegendAlignment</u></a>	Устанавливает выравнивание текста для легенды.
<a href="#"><u>Accumulative</u></a>	Устанавливает флаг накопления значений для серий.
<a href="#"><u>VScaleParams</u></a>	Устанавливает параметры для вертикальной шкалы значений.
<a href="#"><u>DescriptorUpdate</u></a>	Обновляет значение дескриптора серии (по заданной позиции).
<a href="#"><u>ColorUpdate</u></a>	Обновляет цвета серий (по заданной позиции).
<a href="#"><u>ValuesCheck</u></a>	Производит внутренние вычисления для построения графика.
<a href="#"><u>Redraw</u></a>	Перерисовывает график.
<a href="#"><u>DrawBackground</u></a>	Отрисовывает фон.
<a href="#"><u>DrawLegend</u></a>	Перерисовывает легенду.
<a href="#"><u>DrawLegendVertical</u></a>	Отрисовывает вертикальную легенду.
<a href="#"><u>DrawLegendHorizontal</u></a>	Отрисовывает горизонтальную легенду.
<a href="#"><u>CalcScales</u></a>	Рассчитывает координаты шкалы.
<a href="#"><u>DrawScales</u></a>	Перерисовывает все шкалы значений.
<a href="#"><u>DrawScaleLeft</u></a>	Перерисовывает левую шкалу значений.
<a href="#"><u>DrawScaleRight</u></a>	Перерисовывает правую шкалу значений.
<a href="#"><u>DrawScaleTop</u></a>	Перерисовывает верхнюю шкалу значений
<a href="#"><u>DrawScaleBottom</u></a>	Перерисовывает нижнюю шкалу значений.
<a href="#"><u>DrawGrid</u></a>	Перерисовывает сетку.
<a href="#"><u>DrawChart</u></a>	Перерисовывает данные.

**Методы унаследованные от CCanvas**

[CreateBitmap](#), [CreateBitmap](#), [CreateBitmapLabel](#), [CreateBitmapLabel](#), [Attach](#), [Attach](#), [Destroy](#), [ChartObjectName](#), [ResourceName](#), [Width](#), [Height](#), [Update](#), [Resize](#), [Erase](#), [PixelGet](#), [PixelSet](#), [LineVertical](#), [LineHorizontal](#), [Line](#), [Polyline](#), [Polygon](#), [Rectangle](#), [Triangle](#), [Circle](#), [Ellipse](#), [Arc](#), [Arc](#), [Arc](#), [Pie](#), [Pie](#), [FillRectangle](#), [FillTriangle](#), [FillPolygon](#), [FillCircle](#), [FillEllipse](#), [Fill](#), [Fill](#), [PixelSetAA](#), [LineAA](#), [PolylineAA](#), [PolygonAA](#), [TriangleAA](#), [CircleAA](#), [EllipseAA](#), [LineWu](#), [PolylineWu](#), [PolygonWu](#), [TriangleWu](#), [CircleWu](#), [EllipseWu](#), [LineThickVertical](#), [LineThickHorizontal](#), [LineThick](#), [PolylineThick](#), [PolygonThick](#), [PolylineSmooth](#), [PolygonSmooth](#), [FontSet](#), [FontNameSet](#), [FontSizeSet](#), [FontFlagsSet](#), [FontAngleSet](#), [FontGet](#), [FontNameGet](#), [FontSizeGet](#), [FontFlagsGet](#), [FontAngleGet](#), [TextOut](#), [TextWidth](#), [TextHeight](#), [TextSize](#), [GetDefaultColor](#), [TransparentLevelSet](#), [LoadFromFile](#), [LineStyleGet](#), [LineStyleSet](#)

## ColorBackground (метод Get)

Возвращает цвет фона.

```
uint ColorBackground()
```

### Возвращаемое значение

Цвет фона.

## ColorBackground (метод Set)

Устанавливает цвет заднего фона.

```
void ColorBackground(  
    const uint value, // цвет фона  
)
```

### Параметры

*value*

[in] Цвет фона.

## ColorBorder (метод Get)

Возвращает цвет границ.

```
uint ColorBorder()
```

### Возвращаемое значение

Цвет границ.

## ColorBorder (метод Set)

Устанавливает цвет границ.

```
void ColorBorder(  
    const uint value, // цвет границ  
)
```

### Параметры

*value*

[in] Цвет границ.

## ColorText (метод Get)

Возвращает цвет текста.

```
uint ColorText()
```

### Возвращаемое значение

Цвет текста.

## ColorText (метод Set)

Устанавливает цвет текста.

```
void ColorText(  
    const uint value, // цвет текста  
)
```

### Параметры

*value*

[in] Цвет текста.

## ColorGrid (метод Get)

Возвращает цвет сетки.

```
uint ColorGrid()
```

### Возвращаемое значение

Цвет сетки.

## ColorGrid (метод Set)

Устанавливает цвет сетки.

```
void ColorGrid(  
    const uint value, // цвет сетки  
)
```

### Параметры

*value*

[in] Цвет сетки.

## MaxData (метод Get)

Возвращает максимальное количество допустимых данных (серий).

```
uint MaxData()
```

### Возвращаемое значение

Максимальное количество данных (серий).

## MaxData (метод Set)

Устанавливает максимальное количество допустимых данных (серий).

```
void MaxData(  
    const uint value, // количество данных  
)
```

### Параметры

*value*

[in] Максимальное количество данных (серий).

## MaxDescrLen (метод Get)

Возвращает максимальную длину дескрипторов.

```
uint MaxDescrLen()
```

### Возвращаемое значение

Значение максимальной длины дескрипторов.

## MaxDescrLen (метод Set)

Устанавливает максимальную длину дескрипторов.

```
void MaxDescrLen(  
    const uint value, // максимальная длина  
)
```

### Параметры

*value*

[in] Значение максимальной длины дескрипторов.

## ShowFlags (метод Get)

Возвращает флаг видимости элементов графика.

```
bool ShowFlags()
```

### Возвращаемое значение

Значение флага видимости элементов графика.

## ShowFlags (метод Set)

Устанавливает флаг видимости элементов графика.

```
void ShowFlags(  
    const uint flags, // флаг  
)
```

### Parameters

*flags*

[in] Значение флага видимости элементов графика.

## IsShowLegend

Возвращает флаг видимости легенды на графике.

```
bool IsShowLegend()
```

### Возвращаемое значение

true, если легенда видна, иначе false.

## IsShowScaleLeft

Возвращает флаг видимости шкалы значений слева.

```
bool IsShowScaleLeft()
```

### Возвращаемое значение

true, если шкала значений видна, иначе – false.

## IsShowScaleRight

Возвращает флаг видимости шкалы значений справа.

```
bool IsShowScaleRight()
```

### Возвращаемое значение

true, если шкала значений видна, иначе – false.

## IsShowScaleTop

Возвращает флаг видимости шкалы значений сверху.

```
bool IsShowScaleTop()
```

### Возвращаемое значение

true, если шкала значений видна, иначе – false.

## IsShowScaleBottom

Возвращает флаг видимости шкалы значений снизу.

```
bool IsShowScaleBottom()
```

### Возвращаемое значение

true, если шкала значений видна, иначе – false.

## IsShowGrid

Возвращает флаг видимости сетки на графике.

```
bool IsShowGrid()
```

### Возвращаемое значение

true, если сетка видна, иначе – false.

## IsShowDescriptors

Возвращает флаг видимости дескрипторов на графике.

```
bool IsShowDescriptors()
```

### Возвращаемое значение

true, если дескрипторы видны, иначе – false.

## IsShowPercent

Возвращает флаг видимости процентов на графике.

```
bool IsShowPercent()
```

### Возвращаемое значение

true, если проценты видны, иначе – false.

## VScaleMin (Метод Get)

Возвращает минимум на вертикальной шкале значений.

```
double VScaleMin()
```

### Возвращаемое значение

Минимальное значение на вертикальной шкале.

## VScaleMin (Метод Set)

Устанавливает минимум на вертикальной шкале значений.

```
void VScaleMin(  
    const double value, // значение на вертикальной шкале  
)
```

### Параметры

*value*

[in] Минимальное значение.

## VScaleMax

Возвращает максимум на вертикальной шкале значений.

```
double VScaleMax()
```

### Возвращаемое значение

Максимальное значение на вертикальной шкале.

## VScaleMax

Устанавливает максимум на вертикальной шкале значений.

```
void VScaleMax(  
    const double value, // значение на вертикальной шкале  
)
```

### Параметры

*value*

[in] Максимальное значение.

## NumGrid

Возвращает количество вертикальных делений шкалы при построении сетки графика.

```
uint NumGrid()
```

### Возвращаемое значение

Количество делений.

## NumGrid

Устанавливает количество вертикальных делений шкалы при построении сетки графика.

```
void NumGrid(  
    const uint value, // количество делений  
)
```

### Параметры

*value*

[in] Количество делений.

## DataOffset

Возвращает значение смещения данных.

```
int DataOffset()
```

### Возвращаемое значение

Смещение данных.

## DataOffset

Устанавливает значение смещения данных.

```
void DataOffset(
    const int value, // значение
)
```

### Параметры

*value*

[in] Смещение данных.

## DataTotal

Возвращает общее количество серий данных на графике.

```
uint DataTotal()
```

### Возвращаемое значение

Количество серий.

## DrawDescriptors

Виртуальный метод для отрисовки дескрипторов.

```
virtual void DrawDescriptors()
```

## DrawData

Виртуальный метод для отрисовки серии данных по указанному индексу.

```
virtual void DrawData(  
    const uint idx=0, // индекс  
)
```

### Параметры

*idx=0*

[in] Индекс серии.

## Create

Виртуальный метод, который создаёт графический ресурс.

```
virtual bool Create(
    const string      name,           // имя ресурса
    const int         width,          // ширина
    const int         height,         // высота
    ENUM_COLOR_FORMAT clrfmt,        // формат
)
```

### Параметры

*name*

[in] Основание для имени графического ресурса. Имя ресурса формируется при создании путем добавления псевдослучайной строки.

*width*

[in] Ширина (размер по оси X) в пикселях.

*height*

[in] Высота (размер по оси Y) в пикселях.

*clrfmt*

[in] Способ обработки цвета. Более подробно о способах обработки цвета смотрите в описании функции ResourceCreate().

### Возвращаемое значение

true в случае удачи, иначе – false.

## AllowedShowFlags

Устанавливает набор разрешенных флагов видимости для элементов графика.

```
void AllowedShowFlags(  
    const uint flags, // флаги  
)
```

### Параметры

*flags*

[in] Разрешенные флаги.

## ShowLegend

Устанавливает значение флага видимости для легенды (FLAG\_SHOW\_LEGEND).

```
void ShowLegend(  
    const bool flag, // значение флага  
)
```

### Параметры

*flag*

[in] Значение флага:

- true – легенда становится видимой.
- false – легенда становится невидимой.

## ShowScaleLeft

Устанавливает значение флага видимости для левой шкалы (FLAG\_SHOW\_SCALE\_LEFT).

```
void ShowScaleLeft(  
    const bool flag, // значение флага  
)
```

### Параметры

*flag*

[in] Значение флага:

- true – левая шкала становится видимой.
- false – левая шкала становится невидимой.

## ShowScaleRight

Устанавливает значение флага видимости для правой шкалы (FLAG\_SHOW\_SCALE\_RIGHT).

```
void ShowScaleRight(  
    const bool flag, // значение флага  
)
```

### Параметры

*flag*

[in] Значение флага:

- true – правая шкала становится видимой.
- false – правая шкала становится невидимой.

## ShowScaleTop

Устанавливает значение флага видимости для верхней шкалы (FLAG\_SHOW\_SCALE\_TOP).

```
void ShowScaleTop(  
    const bool flag, // значение флага  
)
```

### Параметры

*flag*

[in] Значение флага:

- true — верхняя шкала становится видимой.
- false — верхняя шкала становится невидимой.

## ShowScaleBottom

Устанавливает значение флага видимости для нижней шкалы (FLAG\_SHOW\_SCALE\_BOTTOM).

```
void ShowScaleBottom(  
    const bool flag,           // значение флага  
)
```

### Параметры

*flag*

[in] Значение флага:

- true — нижняя шкала становится видимой.
- false — нижняя шкала становится невидимой.

## ShowGrid

Устанавливает значение флага видимости для сетки (FLAG\_SHOW\_GRID).

```
void ShowGrid(
    const bool flag, // значение флага
)
```

### Параметры

*flag*

[in] Значение флага:

- true – сетка становится видимой.
- false – сетка становится невидимой.

## ShowDescriptors

Устанавливает значение флага видимости для дескрипторов (FLAG\_SHOW\_DESCRIPTOR).

```
void ShowDescriptors(
    const bool flag, // значение флага
)
```

### Параметры

*flag*

[in] Значение флага:

- true – дескриптор становится видимым.
- false – дескриптор становится невидимым.

## ShowValue

Устанавливает флаг видимости для значений (FLAG\_SHOW\_VALUE).

```
void ShowValue(
    const bool flag, // значение флага
)
```

### Параметры

*flag*

[in] Значение флага:

- true — значение становится видимым.
- false — значение становится невидимым.

## ShowPercent

Устанавливает значение флага видимости для процентов (FLAG\_SHOW\_PERCENT).

```
void ShowPercent(  
    const bool flag, // значение флага  
)
```

### Параметры

*flag*

[in] Значение флага:

- true – процент становится видимым.
- false – процент становится невидимым.

## LegendAlignment

Устанавливает выравнивание текста для легенды.

```
void LegendAlignment(
    const ENUM_ALIGNMENT value, // флаг
)
```

### Параметры

*value*

[in] Принимает одно из значений перечисления ENUM\_ALIGNMENT:

- ALIGNMENT\_LEFT – выравнивание по левому краю.
- ALIGNMENT\_TOP – выравнивание по верхнему краю.
- ALIGNMENT\_RIGHT – выравнивание по правому краю.
- ALIGNMENT\_BOTTOM – выравнивание по нижнему краю.

## Accumulative

Устанавливает флаг накопления значений для серий.

```
void Accumulative(
    const bool flag=true, // значение флага
)
```

### Параметры

*flag=true*

[in] Значение флага:

- true — текущее значение серии заменяется суммой всех предыдущих.
- false — стандартный режим отрисовки серий.

## VScaleParams

Устанавливает параметры для вертикальной шкалы значений.

```
void VScaleParams(
    const double max, // максимум
    const double min, // минимум
    const uint grid, // количество делений
)
```

### Параметры

*max*

[in] Минимальное значение.

*min*

[in] Максимальное значение.

*grid*

[in] Количество делений шкалы.

## DescriptorUpdate

Обновляет значение дескриптора серии (по заданной позиции).

```
bool DescriptorUpdate(
    const uint    pos,      // индекс
    const string  descr,   // значение
)
```

### Параметры

*pos*

[in] Индекс серии – порядковый номер её добавления, начиная с 0.

*descr*

[in] Значение дескриптора.

### Возвращаемое значение

true в случае успеха, иначе – false.

## ColorUpdate

Обновляет цвета серии (по заданной позиции).

```
bool ColorUpdate(
    const uint pos, // индекс
    const uint clr, // цвет
)
```

### Параметры

*pos*

[in] Индекс серии – порядковый номер её добавления, начиная с 0.

*clr*

[in] Значение цвета.

### Возвращаемое значение

true в случае успеха, иначе – false.

## ValuesCheck

Вспомогательный виртуальный метод, производит внутренние вычисления для построения графика.

```
virtual void ValuesCheck()
```

## Redraw

Виртуальный метод для перерисовывания графика.

```
virtual void Redraw()
```

## DrawBackground

Виртуальный метод для перерисовки фона.

```
virtual void DrawBackground()
```

## DrawLegend

Виртуальный метод для перерисовки легенды.

```
virtual void DrawLegend()
```

## DrawLegendVertical

Рисует вертикальную легенду.

```
int DrawLegendVertical(
    const int w, // ширина
    const int h, // высота
)
```

### Параметры

*w*

[in] Максимальная ширина текста в легенде.

*h*

[in] Максимальная высота текста в легенде.

### Возвращаемое значение

Ширина легенды в пикселях.

## DrawLegendHorizontal

Рисует горизонтальную легенду.

```
int DrawLegendHorizontal(
    const int w, // 
    const int h, // 
)
```

### Параметры

*w*

[in] Максимальная ширина текста в легенде.

*h*

[in] Максимальная высота текста в легенде.

### Возвращаемое значение

Высота легенды в пикселях.

## CalcScales

Виртуальный метод для расчета координат надписей для шкалы значений.

```
virtual void CalcScales()
```

## DrawScales

Виртуальный метод для перерисовки всех шкал значений.

```
virtual void DrawScales()
```

## DrawScaleLeft

Виртуальный метод для перерисовки левой шкалы значений.

```
virtual int DrawScaleLeft(
    const bool draw, // флаг
)
```

### Параметры

*draw*

[in] Флаг, указывающий, нужно ли выполнять перерисовку шкалы.

### Возвращаемое значение

Ширина шкалы значений.

## DrawScaleRight

Виртуальный метод для перерисовки правой шкалы значений.

```
virtual int DrawScaleRight(
    const bool draw, // флаг
)
```

### Параметры

*draw*

[in] Флаг, указывающий, нужно ли выполнять перерисовку шкалы.

### Возвращаемое значение

Ширина шкалы значений.

## DrawScaleTop

Виртуальный метод для перерисовки верхней шкалы значений.

```
virtual int DrawScaleTop(  
    const bool draw, // флаг  
)
```

### Параметры

*draw*

[in] Флаг, указывающий, нужно ли выполнять перерисовку шкалы.

### Возвращаемое значение

Высота шкалы значений.

## DrawScaleBottom

Виртуальный метод для перерисовки нижней шкалы значений.

```
virtual int DrawScaleBottom(
    const bool draw, // флаг
)
```

### Параметры

*draw*

[in] Флаг, указывающий, нужно ли выполнять перерисовку шкалы.

### Возвращаемое значение

Высота шкалы значений.

## DrawGrid

Виртуальный метод для перерисовки сетки.

```
virtual void DrawGrid()
```

## DrawChart

Виртуальный метод для перерисовки графика.

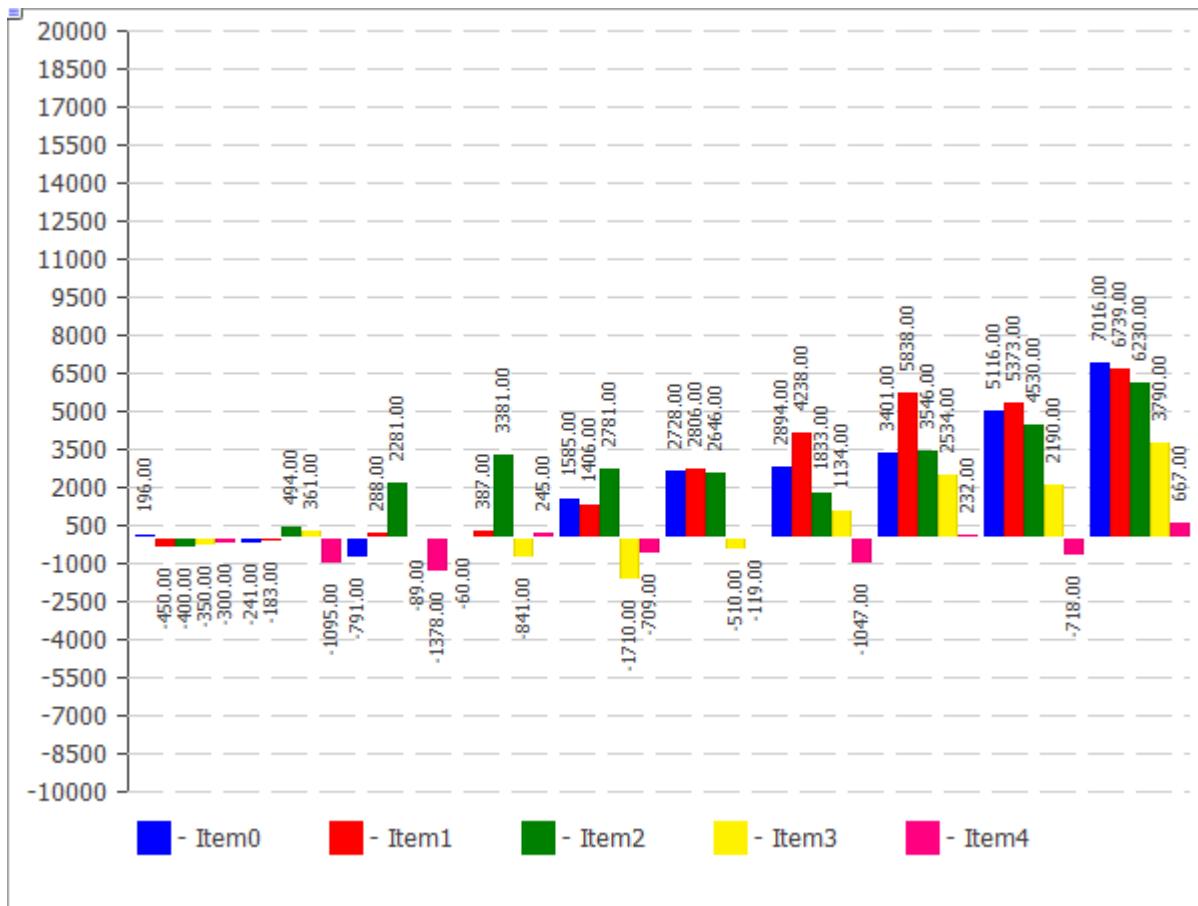
```
virtual void DrawChart()
```

## CHistogramChart

Класс для построения гистограмм.

### Описание

В этом классе реализованы все методы работы с построением столбчатых гистограмм. С их помощью задается ширина столбцов, настраивается работа с сериями данных. Включены методы для работы с градиентной заливкой столбцов гистограмм, что позволяет более наглядно отображать картину данных.



Код вышеприведенного рисунка представлен [ниже](#).

### Декларация

```
class CHistogramChart : public CChartCanvas
```

### Заголовок

```
#include <Canvas\Charts\HistogramChart.mqh>
```

### Иерархия наследования

```
CCanvas
CChartCanvas
CHistogramChart
```

## Методы класса

Метод	Действие
<a href="#">Gradient</a>	Устанавливает флаг с указанием на то, будет ли применяться градиентная заливка столбцов гистограммы.
<a href="#">BarGap</a>	Устанавливает значение отступа гистограммы от начала координат.
<a href="#">BarMinSize</a>	Устанавливает минимальную ширину столбцов гистограммы.
<a href="#">BarBorder</a>	Устанавливает флаг, указывающий на необходимость отрисовки границы для каждого столбца.
<a href="#">Create</a>	Виртуальный метод, который создаёт графический ресурс.
<a href="#">SeriesAdd</a>	Добавляет новую серию данных.
<a href="#">SeriesInsert</a>	Вставляет серию данных на график.
<a href="#">SeriesUpdate</a>	Обновляет серию данных на графике.
<a href="#">SeriesDelete</a>	Удаляет серию данных на графике.
<a href="#">ValueUpdate</a>	Обновляет значение элемента в указанной серии.
<a href="#">DrawData</a>	Виртуальный метод, который рисует гистограмму для заданной серии.
<a href="#">DrawBar</a>	Рисует столбец гистограммы в виде закрашенного прямоугольника.
<a href="#">GradientBrush</a>	Создает кисть для градиентной заливки.

### Методы унаследованные от CCanvas

[CreateBitmap](#), [CreateBitmap](#), [CreateBitmapLabel](#), [CreateBitmapLabel](#), [Attach](#), [Attach](#), [Destroy](#), [ChartObjectName](#), [ResourceName](#), [Width](#), [Height](#), [Update](#), [Resize](#), [Erase](#), [PixelGet](#), [PixelSet](#), [LineVertical](#), [LineHorizontal](#), [Line](#), [Polyline](#), [Polygon](#), [Rectangle](#), [Triangle](#), [Circle](#), [Ellipse](#), [Arc](#), [Arc](#), [Pie](#), [Pie](#), [FillRectangle](#), [FillTriangle](#), [FillPolygon](#), [FillCircle](#), [FillEllipse](#), [Fill](#), [Fill](#), [PixelSetAA](#), [LineAA](#), [PolylineAA](#), [PolygonAA](#), [TriangleAA](#), [CircleAA](#), [EllipseAA](#), [LineWu](#), [PolylineWu](#), [PolygonWu](#), [TriangleWu](#), [CircleWu](#), [EllipseWu](#), [LineThickVertical](#), [LineThickHorizontal](#), [LineThick](#), [PolylineThick](#), [PolygonThick](#), [PolylineSmooth](#), [PolygonSmooth](#), [FontSet](#), [FontNameSet](#), [FontSizeSet](#), [FontFlagsSet](#), [FontAngleSet](#), [FontGet](#), [FontNameGet](#), [FontSizeGet](#), [FontFlagsGet](#), [FontAngleGet](#), [TextOut](#), [TextWidth](#), [TextHeight](#), [TextSize](#), [GetDefaultColor](#), [TransparentLevelSet](#), [LoadFromFile](#), [LineStyleGet](#), [LineStyleSet](#)

### Методы унаследованные от CChartCanvas

[ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [ColorText](#), [ColorText](#), [ColorGrid](#), [ColorGrid](#), [MaxData](#), [MaxData](#), [MaxDescrLen](#), [MaxDescrLen](#), [AllowedShowFlags](#), [ShowFlags](#), [ShowFlags](#), [IsShowLegend](#), [IsShowScaleLeft](#), [IsShowScaleRight](#), [IsShowScaleTop](#), [IsShowScaleBottom](#), [IsShowGrid](#), [IsShowDescriptors](#), [IsShowPercent](#), [ShowLegend](#), [ShowScaleLeft](#), [ShowScaleRight](#), [ShowScaleTop](#), [ShowScaleBottom](#), [ShowGrid](#), [ShowDescriptors](#), [ShowValue](#), [ShowPercent](#), [LegendAlignment](#), [Accumulative](#), [VScaleMin](#), [VScaleMin](#), [VScaleMax](#), [VScaleMax](#), [NumGrid](#), [NumGrid](#), [VScaleParams](#), [DataOffset](#), [DataOffset](#), [DataTotal](#), [DescriptorUpdate](#), [ColorUpdate](#)

Пример

```

//+-----+
//|                               HistogramChartSample.mq5 |
//|           Copyright 2009-2017, MetaQuotes Software Corp. |
//|                               http://www.mql5.com |
//+-----+
#property copyright "2009-2017, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property description "Example of using histogram"
//---
#include <Canvas\Charts\HistogramChart.mqh>
//+-----+
//| inputs
//+-----+


```

```
//--- finish  
chart.Destroy();  
return(0);  
}
```

## Gradient

Устанавливает флаг с указанием на то, будет ли применяться градиентная заливка столбцов гистограммы.

```
void Gradient(
    const bool flag=true, // значение флага
)
```

### Параметры

*flag=true*

Значение флага: true, если включена градиентная заливка, иначе – false.

## BarGap

Устанавливает значение отступа гистограммы от начала координат.

```
void BarGap(
    const uint value, // отступ
)
```

### Параметры

*value*

[in] Значение отступа гистограммы.

## BarMinSize

Устанавливает минимальную ширину столбцов гистограммы.

```
void BarMinSize(
    const uint value, // минимальная ширина
)
```

### Параметры

*value*

[in] Минимальная ширина.

## BarBorder

Устанавливает флаг, указывающий на необходимость отрисовки границы для каждого столбца.

```
void BarBorder(
    const uint value, // флаг
)
```

### Параметры

*value*

[in] Значение флага:

- true – границы будут отрисовываться
- false – границы не будут отрисовываться

## Create

Виртуальный метод, который создаёт графический ресурс.

```
virtual bool Create(
    const string      name,      // имя
    const int         width,     // ширина
    const int         height,    // высота
    ENUM_COLOR_FORMAT clrfmt,   // формат
)
```

### Параметры

*name*

[in] Основание для имени графического ресурса. Имя ресурса формируется при создании путем добавления псевдослучайной строки.

*width*

[in] Ширина (размер по оси X) в пикселях.

*height*

[in] Высота (размер по оси Y) в пикселях.

*clrfmt*

[in] Способ обработки цвета. Более подробно о способах обработки цвета смотрите в описании функции ResourceCreate().

### Возвращаемое значение

true в случае удачи, иначе – false.

## SeriesAdd

Добавляет новую серию данных.

```
bool SeriesAdd(
    const double& value[], // значения
    const string descr,   // подпись
    const uint    clr,    // цвет
)
```

### Параметры

*value[]*

[in] Серия данных.

*descr*

[in] Подпись серии.

*clr*

[in] Цвет отображения серии.

### Возвращаемое значение

true в случае успеха, иначе – false.

## SeriesInsert

Вставляет серию данных на график.

```
bool SeriesInsert(
    const uint    pos,          // индекс
    const double& value[],     // значения
    const string  descr,        // подпись
    const uint    clr,          // цвет
)
```

### Параметры

*pos*

[in] Индекс для вставки.

*value[]*

[in] Серия данных.

*descr*

[in] Подпись серии.

*clr*

[in] Цвет отображения серии.

### Возвращаемое значение

true в случае успеха, иначе – false.

## SeriesUpdate

Обновляет серию данных на графике.

```
bool SeriesUpdate(
    const uint    pos,          // индекс
    const double &value[],     // значения
    const string descr,        // подпись
    const uint    clr,          // цвет
)
```

### Параметры

*pos*

[in] Индекс серии — порядковый номер её добавления, начиная с 0.

*&value[]*

[in] Новые значения для серии данных.

*descr*

[in] Подпись серии.

*clr*

[in] Цвет отображения серии.

### Возвращаемое значение

true - в случае успеха, иначе false.

## SeriesDelete

Удаляет серию данных на графике.

```
bool SeriesDelete(
    const uint pos, // индекс
)
```

### Параметры

*pos*

[in] Индекс серии — порядковый номер её добавления, начинная с 0.

### Возвращаемое значение

true в случае успеха, иначе — false.

## ValueUpdate

Обновляет указанное значение в указанной серии.

```
bool ValueUpdate(
    const uint series, // индекс серии
    const uint pos,    // индекс элемента
    double     value, // значение
)
```

### Параметры

*series*

[in] Индекс серии – порядковый номер её добавления, начиная с 0.

*pos*

[in] Индекс элемента в серии.

*value*

[in] Новое значение.

### Возвращаемое значение

true в случае успеха, иначе – false.

## DrawData

Виртуальный метод, который рисует гистограмму для заданной серии.

```
virtual void DrawData(
    const uint index, // индекс
)
```

### Параметры

*index*

[in] Индекс серии — порядковый номер её добавления, начиная с 0.

## DrawBar

Рисует столбец гистограммы в виде закрашенного прямоугольника.

```
void DrawBar(
    const int    x,      // координата x
    const int    y,      // координата y
    const int    w,      // ширина
    const int    h,      // высота
    const uint   clr,    // цвет
)
```

### Параметры

*x*

[in] Координата X левой верхней точки прямоугольника.

*y*

[in] Координата Y левой верхней точки прямоугольника.

*w*

[in] Ширина прямоугольника.

*h*

[in] Высота прямоугольника.

*clr*

[in] Цвет прямоугольника.

## GradientBrush

Создает кисть для градиентной заливки.

```
void GradientBrush(  
    const int    size,          // размер  
    const uint   fill_clr,     // цвет заливки  
)
```

### Параметры

*size*

[in] Толщина кисти

*fill\_clr*

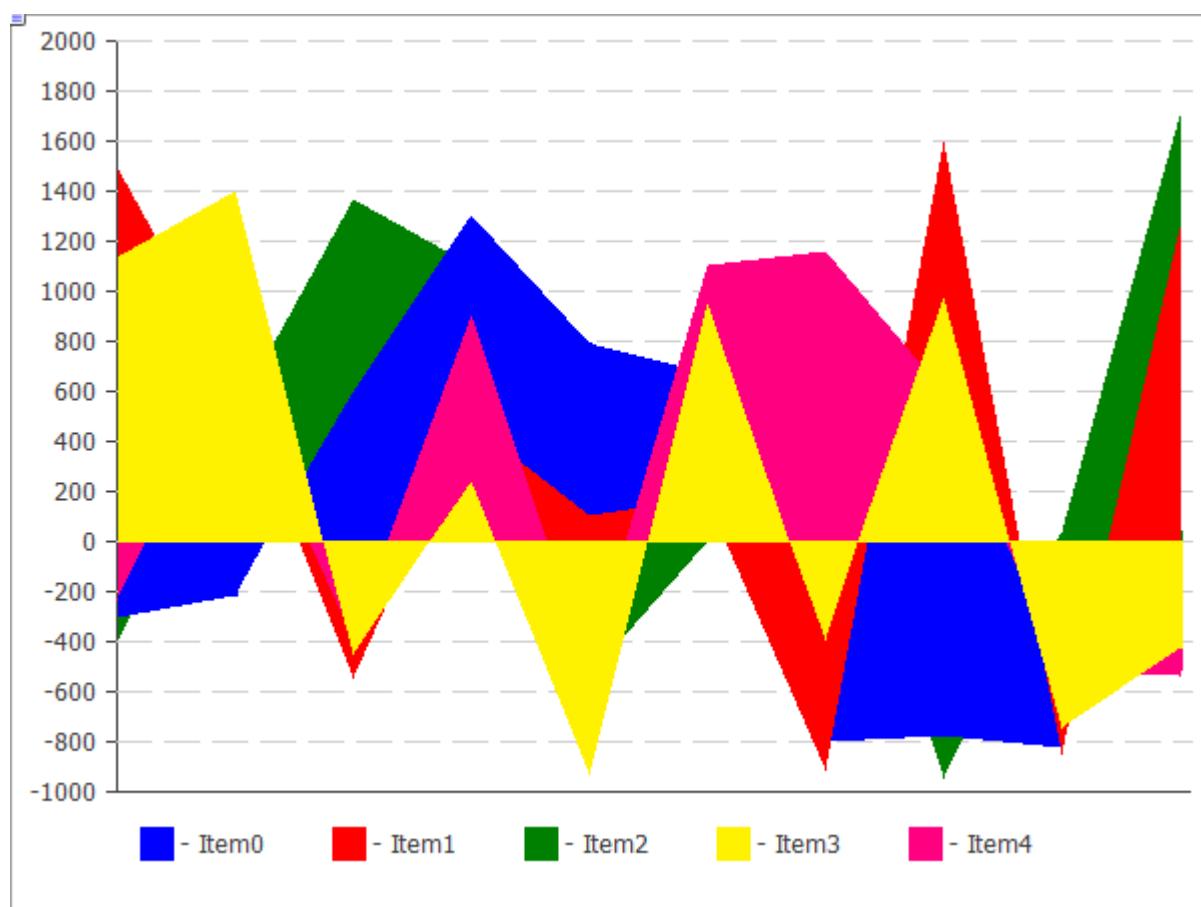
[in] Цвет заливки

## CLineChart

Класс для построения кривых.

### Описание

Методы, входящие в класс, предназначены для работы с кривыми на графике. Включена возможность заливки области, ограниченной построенной кривой.



Код вышеприведенного рисунка представлен [ниже](#).

### Декларация

```
class CLineChart : public CChartCanvas
```

### Заголовок

```
#include <Canvas\Charts\LineChart.mqh>
```

### Иерархия наследования

```
CCanvas  
CChartCanvas  
CLineChart
```

### Методы класса

Method	Action
<a href="#">Filled</a>	Устанавливает флаг заливки области под кривой, заданной серией данных.
<a href="#">Create</a>	Создает графический ресурс.
<a href="#">SeriesAdd</a>	Добавляет новую серию данных.
<a href="#">SeriesInsert</a>	Вставляет серию данных на график.
<a href="#">SeriesUpdate</a>	Обновляет серию данных на графике.
<a href="#">SeriesDelete</a>	Удаляет серию данных с графика.
<a href="#">ValueUpdate</a>	Обновляет указанное значение в указанной серии.
<a href="#">DrawChart</a>	Виртуальный метод, который выполняет отрисовку кривой и всех её элементов.
<a href="#">DrawData</a>	Виртуальный метод, который рисует кривую для заданной серии.
<a href="#">CalcArea</a>	Рассчитывает площадь под кривой, заданной серией данных.

#### Методы унаследованные от CCanvas

[CreateBitmap](#), [CreateBitmap](#), [CreateBitmapLabel](#), [CreateBitmapLabel](#), [Attach](#), [Attach](#), [Destroy](#), [ChartObjectName](#), [ResourceName](#), [Width](#), [Height](#), [Update](#), [Resize](#), [Erase](#), [PixelGet](#), [PixelSet](#), [LineVertical](#), [LineHorizontal](#), [Line](#), [Polyline](#), [Polygon](#), [Rectangle](#), [Triangle](#), [Circle](#), [Ellipse](#), [Arc](#), [Arc](#), [Arc](#), [Pie](#), [Pie](#), [FillRectangle](#), [FillTriangle](#), [FillPolygon](#), [FillCircle](#), [FillEllipse](#), [Fill](#), [Fill](#), [PixelSetAA](#), [LineAA](#), [PolylineAA](#), [PolygonAA](#), [TriangleAA](#), [CircleAA](#), [EllipseAA](#), [LineWu](#), [PolylineWu](#), [PolygonWu](#), [TriangleWu](#), [CircleWu](#), [EllipseWu](#), [LineThickVertical](#), [LineThickHorizontal](#), [LineThick](#), [PolylineThick](#), [PolygonThick](#), [PolylineSmooth](#), [PolygonSmooth](#), [FontSet](#), [FontNameSet](#), [FontSizeSet](#), [FontFlagsSet](#), [FontAngleSet](#), [FontGet](#), [FontNameGet](#), [FontSizeGet](#), [FontFlagsGet](#), [FontAngleGet](#), [TextOut](#), [TextWidth](#), [TextHeight](#), [TextSize](#), [GetDefaultColor](#), [TransparentLevelSet](#), [LoadFromFile](#), [LineStyleGet](#), [LineStyleSet](#)

#### Методы унаследованные от CChartCanvas

[ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [ColorText](#), [ColorText](#), [ColorGrid](#), [ColorGrid](#), [MaxData](#), [MaxData](#), [MaxDescrLen](#), [MaxDescrLen](#), [AllowedShowFlags](#), [ShowFlags](#), [ShowFlags](#), [IsShowLegend](#), [IsShowScaleLeft](#), [IsShowScaleRight](#), [IsShowScaleTop](#), [IsShowScaleBottom](#), [IsShowGrid](#), [IsShowDescriptors](#), [IsShowPercent](#), [ShowLegend](#), [ShowScaleLeft](#), [ShowScaleRight](#), [ShowScaleTop](#), [ShowScaleBottom](#), [ShowGrid](#), [ShowDescriptors](#), [ShowValue](#), [ShowPercent](#), [LegendAlignment](#), [Accumulative](#), [VScaleMin](#), [VScaleMin](#), [VScaleMax](#), [VScaleMax](#), [NumGrid](#), [NumGrid](#), [VScaleParams](#), [DataOffset](#), [DataOffset](#), [DataTotal](#), [DescriptorUpdate](#), [ColorUpdate](#)

#### Пример

```

//+-----+
//|                               LineChartSample.mq5 |
//|           Copyright 2009-2017, MetaQuotes Software Corp. |
//|                               http://www.mql5.com |
//+-----+
#property copyright "2009-2017, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property description "Example of using line chart"
//---
#include <Canvas\Charts\LineChart.mqh>
//+-----+
//| inputs
//+-----+


```

```
//--- finish  
chart.Destroy();  
return(0);  
}
```

## Filled

Устанавливает флаг с указанием, нужно ли закрашивать область под кривой, заданной серией данных.

```
void Filled(  
    const bool flag=true, // флаг  
)
```

### Параметры

*flag=true*

[in] Значение флага:

- true — закрашивать область под кривой
- false — не закрашивать область под кривой

## Create

Виртуальный метод, который создаёт графический ресурс.

```
virtual bool Create(
    const string      name,      // имя
    const int         width,     // ширина
    const int         height,    // высота
    ENUM_COLOR_FORMAT clrfmt,   // формат
)
```

### Параметры

*name*

[in] Основание для имени графического ресурса. Имя ресурса формируется при создании путем добавления псевдослучайной строки.

*width*

[in] Ширина (размер по оси X) в пикселях.

*height*

[in] Высота (размер по оси Y) в пикселях.

*clrfmt*

[in] Способ обработки цвета. Более подробно о способах обработки цвета смотрите в описании функции ResourceCreate().

### Возвращаемое значение

true в случае удачи, иначе – false.

## SeriesAdd

Добавляет новую серию данных.

```
bool SeriesAdd(
    const double& value[], // значения
    const string descr,   // подпись
    const uint    clr,    // цвет
)
```

### Параметры

*value[]*

[in] Серия данных.

*descr*

[in] Подпись серии.

*clr*

[in] Цвет отображения серии.

### Возвращаемое значение

true в случае успеха, иначе – false.

## SeriesInsert

Вставляет серию данных на график.

```
bool SeriesInsert(
    const uint      pos,          // индекс
    const double&   value[],     // значения
    const string    descr,        // подпись
    const uint      clr,          // цвет
)
```

### Параметры

*pos*

[in] Индекс для вставки.

*value[]*

[in] Серия данных.

*descr*

[in] Подпись серии.

*clr*

[in] Цвет отображения серии.

### Возвращаемое значение

true в случае успеха, иначе – false.

## SeriesUpdate

Обновляет серию данных на графике.

```
bool SeriesUpdate(
    const uint    pos,          // индекс
    const double& value[],     // значения
    const string  descr,       // подпись
    const uint    clr,          // цвет
)
```

### Параметры

*pos*

[in] Индекс серии – порядковый номер её добавления, начиная с 0.

*value[]*

[in] Новые значения для серии данных.

*descr*

[in] Подпись серии.

*clr*

[in] Цвет отображения серии.

### Возвращаемое значение

true в случае успеха, иначе – false.

## SeriesDelete

Удаляет серию данных с графика.

```
bool SeriesDelete(  
    const uint pos, // индекс  
)
```

### Параметры

*pos*

[in] Индекс серии — порядковый номер её добавления, начиная с 0.

### Возвращаемое значение

true в случае успеха, иначе — false.

## ValueUpdate

Обновляет указанное значение в указанной серии.

```
bool ValueUpdate(
    const uint series, // индекс серии
    const uint pos, // индекс элемента
    double value, // значение
)
```

### Параметры

*series*

[in] Индекс серии – порядковый номер её добавления, начиная с 0.

*pos*

[in] Индекс элемента в серии.

*value*

[in] Новое значение.

### Возвращаемое значение

true в случае успеха, иначе – false.

## DrawChart

Виртуальный метод, который выполняет отрисовку кривой и всех её элементов.

```
virtual void DrawChart()
```

## DrawData

Виртуальный метод, который рисует кривую для заданной серии.

```
virtual void DrawData(
    const uint index, // индекс
)
```

### Parameters

*index*

[in] Индекс серии — порядковый номер её добавления, начиная с 0.

## CalcArea

Рассчитывает площадь под кривой, заданной серией данных.

```
double CalcArea(
    const uint index, // индекс
)
```

### Параметры

*index*

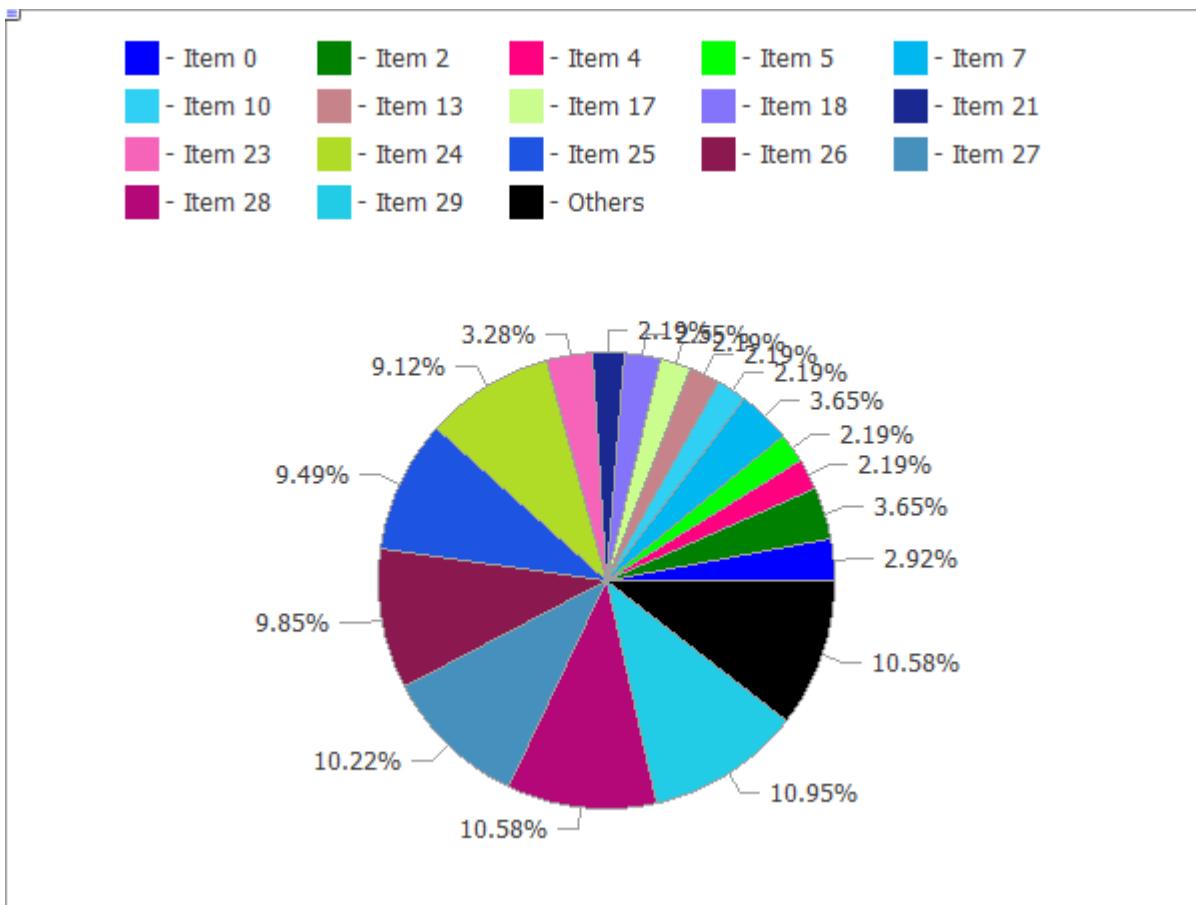
[in] Индекс серии — порядковый номер её добавления, начиная с 0.

### Возвращаемое значение

Площадь фигуры, которую ограничивает кривая, заданная серией данных.

## CPieChart

Класс для построения круговых диаграмм.



Код вышеприведенного рисунка представлен [ниже](#).

### Описание

Методы, входящие в этот класс, предназначены для полноценной работы с круговыми диаграммами, начиная с создания графического ресурса и заканчивая оформлением подписей к сегментам.

### Декларация

```
class CPieChart : public CChartCanvas
```

### Заголовок

```
#include <Canvas\Charts\PieChart.mqh>
```

### Иерархия наследования

```
CCanvas
CChartCanvas
CPieChart
```

## Методы класса

Метод	Действие
<a href="#">Create</a>	Виртуальный метод, который создаёт графический ресурс.
<a href="#">SeriesSet</a>	Задает серию значений, которые будут показаны на диаграмме.
<a href="#">ValueAdd</a>	Добавляет новое значение на диаграмму (в конец).
<a href="#">ValueInsert</a>	Вставляет новое значение на диаграмму (по указанной позиции).
<a href="#">ValueUpdate</a>	Обновляет значение на диаграмме (по указанной позиции).
<a href="#">ValueDelete</a>	Удаляет значение с диаграммы (по заданной позиции).
<a href="#">DrawChart</a>	Виртуальный метод, который выполняет отрисовку диаграммы и всех её элементов.
<a href="#">DrawPie</a>	Рисует сегмент диаграммы, соответствующий определенному значению.
<a href="#">LabelMake</a>	Генерирует подпись сегмента на основе его значения и изначальной подписи.

### Методы унаследованные от CCanvas

[CreateBitmap](#), [CreateBitmap](#), [CreateBitmapLabel](#), [CreateBitmapLabel](#), [Attach](#), [Attach](#), [Destroy](#), [ChartObjectName](#), [ResourceName](#), [Width](#), [Height](#), [Update](#), [Resize](#), [Erase](#), [PixelGet](#), [PixelSet](#), [LineVertical](#), [LineHorizontal](#), [Line](#), [Polyline](#), [Polygon](#), [Rectangle](#), [Triangle](#), [Circle](#), [Ellipse](#), [Arc](#), [Arc](#), [Pie](#), [Pie](#), [FillRectangle](#), [FillTriangle](#), [FillPolygon](#), [FillCircle](#), [FillEllipse](#), [Fill](#), [Fill](#), [PixelSetAA](#), [LineAA](#), [PolylineAA](#), [PolygonAA](#), [TriangleAA](#), [CircleAA](#), [EllipseAA](#), [LineWu](#), [PolylineWu](#), [PolygonWu](#), [TriangleWu](#), [CircleWu](#), [EllipseWu](#), [LineThickVertical](#), [LineThickHorizontal](#), [LineThick](#), [PolylineThick](#), [PolygonThick](#), [PolylineSmooth](#), [PolygonSmooth](#), [FontSet](#), [FontNameSet](#), [FontSizeSet](#), [FontFlagsSet](#), [FontAngleSet](#), [FontGet](#), [FontNameGet](#), [FontSizeGet](#), [FontFlagsGet](#), [FontAngleGet](#), [TextOut](#), [TextWidth](#), [TextHeight](#), [TextSize](#), [GetDefaultColor](#), [TransparentLevelSet](#), [LoadFromFile](#), [LineStyleGet](#), [LineStyleSet](#)

### Методы унаследованные от CChartCanvas

[ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [ColorText](#), [ColorText](#), [ColorGrid](#), [ColorGrid](#), [MaxData](#), [MaxData](#), [MaxDescrLen](#), [MaxDescrLen](#), [AllowedShowFlags](#), [ShowFlags](#), [ShowFlags](#), [IsShowLegend](#), [IsShowScaleLeft](#), [IsShowScaleRight](#), [IsShowScaleTop](#), [IsShowScaleBottom](#), [IsShowGrid](#), [IsShowDescriptors](#), [IsShowPercent](#), [ShowLegend](#), [ShowScaleLeft](#), [ShowScaleRight](#), [ShowScaleTop](#), [ShowScaleBottom](#), [ShowGrid](#), [ShowDescriptors](#), [ShowValue](#), [ShowPercent](#), [LegendAlignment](#), [Accumulative](#), [VScaleMin](#), [VScaleMin](#), [VScaleMax](#), [VScaleMax](#), [NumGrid](#), [NumGrid](#), [VScaleParams](#), [DataOffset](#), [DataOffset](#), [DataTotal](#), [DescriptorUpdate](#), [ColorUpdate](#)

## Пример

```
//-----+  
//| Copyright 2009-2017, MetaQuotes Software Corp. |  
//| http://www.mql5.com |  
//-----+  
#property copyright "2009-2017, MetaQuotes Software Corp."  
#property link "http://www.mql5.com"  
#property description "Example of using pie chart"  
//---  
#include <Canvas\Charts\PieChart.mqh>  
//-----+  
//| inputs  
//-----+  
input int Width=600;  
input int Height=450;  
//-----+  
//| Script program start function  
//-----+  
int OnStart(void)  
{  
//--- check  
    if(Width<=0 || Height<=0)  
    {  
        Print("Too simple.");  
        return(-1);  
    }  
//--- create chart  
    CPieChart pie_chart;  
    if(!pie_chart.CreateBitmapLabel("PieChart",10,10,Width,Height))  
    {  
        Print("Error creating pie chart: ",GetLastError());  
        return(-1);  
    }  
    pie_chart.ShowPercent();  
//--- draw  
    for(uint i=0;i<30;i++)  
    {  
        pie_chart.ValueAdd(100*(i+1),"Item "+IntegerToString(i));  
        Sleep(10);  
    }  
    Sleep(2000);  
//--- disable legend  
    pie_chart.LegendAlignment(ALIGNMENT_LEFT);  
    Sleep(2000);  
//--- disable legend  
    pie_chart.LegendAlignment(ALIGNMENT_RIGHT);
```

```

    Sleep(2000);
//--- disable legend
    pie_chart.LegendAlignment(ALIGNMENT_TOP);
    Sleep(2000);
//--- disable legend
    pie_chart.ShowLegend(false);
    Sleep(2000);
//--- disable percentage
    pie_chart.ShowPercent(false);
    Sleep(2000);
//--- disable descriptors
    pie_chart.ShowDescriptors(false);
    Sleep(2000);
//--- enable all
    pie_chart.ShowLegend();
    pie_chart.ShowValue();
    pie_chart.ShowDescriptors();
    Sleep(2000);
//--- or like this
    pie_chart.ShowFlags(FLAGS_SHOW_LEGEND|FLAGS_SHOW_DESCRIPTOR|FLAGS_SHOW_PERCENT);
    uint total=pie_chart.DataTotal();
//--- play with values
    for(uint i=0;i<total && !IsStopped();i++)
    {
        pie_chart.ValueUpdate(i,100*(rand()%10+1));
        Sleep(1000);
    }
//--- play with colors
    for(uint i=0;i<total && !IsStopped();i++)
    {
        pie_chart.ColorUpdate(i%total,RandomRGB());
        Sleep(1000);
    }
//--- rotate
    while(!IsStopped())
    {
        pie_chart.DataOffset(pie_chart.DataOffset()+1);
        Sleep(200);
    }
//--- finish
    pie_chart.Destroy();
    return(0);
}
//----------------------------------------------------------------------------------------------------------------+
//| Random RGB color |
//----------------------------------------------------------------------------------------------------------------+
uint RandomRGB(void)
{
    return(XRGB(rand()%255,rand()%255,rand()%255));
}

```

{}

## Create

Виртуальный метод, который создаёт графический ресурс.

```
virtual bool Create(
    const string      name,      // имя
    const int         width,     // ширина
    const int         height,    // высота
    ENUM_COLOR_FORMAT clrfmt,   // формат
)
```

### Параметры

*name*

[in] Основание для имени графического ресурса. Имя ресурса формируется при создании путем добавления псевдослучайной строки.

*width*

[in] Ширина (размер по оси X) в пикселях.

*height*

[in] Высота (размер по оси Y) в пикселях.

*clrfmt*

[in] Способ обработки цвета. Более подробно о способах обработки цвета смотрите в описании функции ResourceCreate().

### Возвращаемое значение

true в случае удачи, иначе – false.

## SeriesSet

Задает серию значений, которые будут показаны на диаграмме.

```
bool SeriesSet(
    const double& value[], // значения
    const string& text[], // надписи
    const uint& clr[], // цвета
)
```

### Параметры

*value[]*

[in] Массив значений.

*text[]*

[in] Массив подписей значений.

*clr[]*

[in] Массив цветов значений.

### Возвращаемое значение

true в случае успеха, иначе – false.

## ValueAdd

Добавляет новое значение на диаграмму (в конец).

```
bool ValueAdd(
    const double value, // значение
    const string descr, // подпись
    const uint    clr, // цвет
)
```

### Параметры

*value*

[in] Значение.

*descr*

[in] Подпись значения.

*clr*

[in] Цвет значения.

### Возвращаемое значение

true в случае успеха, иначе – false.

## ValueInsert

Вставляет новое значение на диаграмму (по указанной позиции).

```
bool ValueInsert(
    const uint    pos,      // индекс
    const double  value,    // значение
    const string  descr,    // подпись
    const uint    clr,      // цвет
)
```

### Параметры

*pos*

[in] Индекс для вставки.

*value*

[in] Значение.

*descr*

[in] Подпись значения.

*clr*

[in] Цвет значения.

### Возвращаемое значение

true в случае успеха, иначе – false.

## ValueUpdate

Обновляет значение на диаграмме (по указанной позиции).

```
bool ValueUpdate(
    const uint    pos,      // индекс
    const double  value,    // значение
    const string  descr,    // подпись
    const uint    clr,      // цвет
)
```

### Параметры

*pos*

[in] Индекс значения — порядковый номер его добавления, начиная с 0.

*value*

[in] Значение.

*descr*

[in] Подпись значения.

*clr*

[in] Цвет значения.

### Возвращаемое значение

true в случае успеха, иначе — false.

## ValueDelete

Удаляет значение с диаграммы (по заданной позиции).

```
bool ValueDelete(  
    const uint pos, // индекс  
)
```

### Параметры

*pos*

[in] Индекс значения — порядковый номер его добавления, начиная с 0.

### Возвращаемое значение

true в случае успеха, иначе — false.

## DrawChart

Виртуальный метод, который выполняет отрисовку диаграммы и всех её элементов.

```
virtual void DrawChart()
```

## DrawPie

Рисует сегмент диаграммы, соответствующий определенному значению.

```
void DrawPie(
    double    fi3,    // угол луча из центра диаграммы, задающий первую границу дуги
    double    fi4,    // угол луча из центра диаграммы, задающий вторую границу дуги
    int      idx,    // индекс
    CPoint&  p[],    //
    const uint  clr,    //
)
```

### Параметры

*fi3*

[in] Угол в радианах, задающий первую границу дуги.

*fi4*

[in] Угол в радианах, задающий вторую границу дуги.

*idx*

[in] Индекс значения, которому соответствует сегмент.

*p []*

[in] Массив опорных точек (x, y) для построения сегментов.

*clr*

[in] Цвет сегмента.

## LabelMake

Генерирует подпись сегмента на основе его значения и изначальной подписи.

```
string LabelMake(
    const string  text,      // подпись
    const double   value,     // значение
    const bool     to_left,   // флаг
)
```

### Параметры

*text*

[in] Подпись.

*value*

[in] Значение.

*to\_left*

[in] Определяет порядок компоновки подписи:

- true – подпись, затем значение.
- false – значение, затем подпись.

### Возвращаемое значение

Подпись сегмента.

## CChart

Класс CChart является классом для упрощенного доступа к свойствам графика.

### Описание

Класс CChart обеспечивает доступ к свойствам графика.

### Декларация

```
class CChart : public CObject
```

### Заголовок

```
#include <Charts\Chart.mqh>
```

### Иерархия наследования

[CObject](#)

CChart

### Методы класса по группам

Доступ к защищенным данным	
<a href="#">ChartID</a>	Получает идентификатор графика
<b>Общие свойства</b>	
<a href="#">Mode</a>	Получить/установить тип графика (свечи, бары или линия)
<a href="#">Foreground</a>	Получить/установить флаг "Ценовой график на заднем фоне"
<a href="#">Shift</a>	Получить/установить флаг "Режим отступа ценового графика от правого края"
<a href="#">ShiftSize</a>	Получить/установить размер отступа нулевого бара от правого края в процентах
<a href="#">AutoScroll</a>	Получить/установить флаг "Режим автоматического перехода к правому краю графика"
<a href="#">Scale</a>	Получить/установить масштаб
<a href="#">ScaleFix</a>	Получить/установить флаг "Режим фиксированного масштаба"
<a href="#">ScaleFix_11</a>	Получить/установить флаг "Режим масштаба 1:1"
<a href="#">FixedMax</a>	Получить/установить фиксированный максимум графика

<a href="#">FixedMin</a>	Получить/установить фиксированный минимум графика
<a href="#">ScalePPB</a>	Получить/установить флаг "Режим указания масштаба в пунктах на бар"
<a href="#">PointsPerBar</a>	Получить/установить масштаб в пунктах на бар
<b>Свойства "Show"</b>	
<a href="#">ShowOHLC</a>	Получить/установить флаг "Отображение значений OHLC"
<a href="#">ShowLineBid</a>	Получить/установить флаг "Отображение значения Bid горизонтальной линией"
<a href="#">ShowLineAsk</a>	Получить/установить флаг "Отображение значения Ask горизонтальной линией"
<a href="#">ShowLastLine</a>	Получить/установить флаг "Отображение значения Last горизонтальной линией"
<a href="#">ShowPeriodSep</a>	Получить/установить флаг "Отображение вертикальных разделителей между соседними периодами"
<a href="#">ShowGrid</a>	Получить/установить флаг "Отображение сетки"
<a href="#">ShowVolumes</a>	Получить/установить флаг "Отображение объемов"
<a href="#">ShowObjectDescr</a>	Получить/установить флаг "Отображение всплывающих описаний графических объектов"
<a href="#">ShowDateScale</a>	Установить флаг "Отображение шкалы времени"
<a href="#">ShowPriceScale</a>	Установить флаг "Отображение ценовой шкалы"
<b>Свойства "Color"</b>	
<a href="#">ColorBackground</a>	Получить/установить цвет фона
<a href="#">ColorForeground</a>	Получить/установить цвет осей, шкалы и строки OHLC
<a href="#">ColorGrid</a>	Получить/установить цвет сетки
<a href="#">ColorBarUp</a>	Получить/установить цвет бара вверх, тени и окантовки тела бычьей свечи
<a href="#">ColorBarDown</a>	Получить/установить цвет бара вниз, тени и окантовки тела медвежьей свечи

<a href="#">ColorCandleBull</a>	Получить/установить цвет тела бычьей свечи
<a href="#">ColorCandleBear</a>	Получить/установить цвет тела медвежьей свечи
<a href="#">ColorChartLine</a>	Получить/установить цвет линии графика и японских свечей "Доджи"
<a href="#">ColorVolumes</a>	Получить/установить цвет объемов и уровней открытия позиций
<a href="#">ColorLineBid</a>	Получить/установить цвет линии Bid-цены
<a href="#">ColorLineAsk</a>	Получить/установить цвет линии Ask-цены
<a href="#">ColorLineLast</a>	Получить/установить цвет линии цены последней совершенной сделки
<a href="#">ColorStopLevels</a>	Получить/установить цвет уровней стоп-ордеров (Stop Loss и Take Profit)
<b>Свойства "Read only"</b>	
<a href="#">VisibleBars</a>	Получить количество баров на графике, доступных для отображения
<a href="#">WindowsTotal</a>	Получить общее количество окон графика, включая подокна индикаторов
<a href="#">WindowsVisible</a>	Получить видимость подокон
<a href="#">WindowHandle</a>	Получить хэндл графика (HWND)
<a href="#">FirstVisibleBar</a>	Получить номер первого видимого бара на графике
<a href="#">WidthInBars</a>	Получить ширину графика в барах
<a href="#">WidthInPixels</a>	Получить ширину графика в пикселях
<a href="#">HeightInPixels</a>	Получить высоту окна в пикселях
<a href="#">PriceMin</a>	Получить минимум окна
<a href="#">PriceMax</a>	Получить максимум окна
<b>Свойства</b>	
<a href="#">Attach</a>	Привязывает график к экземпляру класса
<a href="#">FirstChart</a>	Получить идентификатор первого графика
<a href="#">NextChart</a>	Получить идентификатор графика, следующего за указанным
<a href="#">Open</a>	Открыть новый график
<a href="#">Detach</a>	Отвязывает график от экземпляра класса
<a href="#">Close</a>	Закрыть указанный график

<a href="#">BringToFront</a>	Показывает график поверх всех других
<a href="#">EventObjectCreate</a>	Устанавливает флаг отправки сообщений о событиях создания графического объекта
<a href="#">EventObjectDelete</a>	Устанавливает флаг отправки сообщений о событиях удаления графического объекта
<b>Индикаторы</b>	
<a href="#">IndicatorAdd</a>	Добавляет на указанное окно графика индикатор с указанным хэндлом
<a href="#">IndicatorDelete</a>	Удаляет с указанного окна графика индикатор с указанным именем
<a href="#">IndicatorsTotal</a>	Возвращает количество всех индикаторов, присоединенных к указанному окну графика
<a href="#">IndicatorName</a>	Возвращает короткое имя индикатора по номеру в списке индикаторов на указанном окне графика
<b>Навигация</b>	
<a href="#">Navigate</a>	Сдвиг графика
<b>Доступ к API MQL5</b>	
<a href="#">Symbol</a>	Получить имя символа графика
<a href="#">Period</a>	Получить значение периода графика
<a href="#">Redraw</a>	Принудительная перерисовка графика
<a href="#">GetInteger</a>	Получить значение указанного integer-свойства
<a href="#">SetInteger</a>	Установить значение указанного integer-свойства
<a href="#">GetDouble</a>	Получить значение указанного double-свойства
<a href="#">SetDouble</a>	Установить значение указанного double-свойства
<a href="#">GetString</a>	Получить значение указанного string-свойства
<a href="#">SetString</a>	Установить значение указанного string-свойства
<a href="#">SetSymbolPeriod</a>	Установить значения символа и периода указанного графика
<a href="#">ApplyTemplate</a>	Применяет к графику указанный шаблон
<a href="#">ScreenShot</a>	Обеспечивает скриншот указанного графика

<a href="#"><u>WindowOnDropped</u></a>	Получить номер подокна графика, на которое брошен мышкой данный эксперт или скрипт
<a href="#"><u>PriceOnDropped</u></a>	Получить ценовую координату, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт
<a href="#"><u>TimeOnDropped</u></a>	Получить временную координату, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт
<a href="#"><u>XOnDropped</u></a>	Получить координату по оси X, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт
<a href="#"><u>YOnDropped</u></a>	Получить координату по оси Y, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт
<b>Ввод/вывод</b>	
virtual <a href="#"><u>Save</u></a>	Виртуальный метод записи в файл
virtual <a href="#"><u>Load</u></a>	Виртуальный метод чтения из файла
virtual <a href="#"><u>Type</u></a>	Виртуальный метод идентификации

Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

## ChartID

Получает идентификатор графика

```
long ChartID() const
```

### Возвращаемое значение

Идентификатор "привязанного" к экземпляру класса графика. Если нет "привязанного" графика возвращается -1.

## Mode (метод Get)

Получает вид графика (свечи, бары или линия).

```
ENUM_CHART_MODE Mode() const
```

### Возвращаемое значение

Вид графика (свечи, бары или линия, привязанного к экземпляру класса. Если нет "привязанного" графика возвращается [WRONG\\_VALUE](#).

## Mode (метод Set)

Устанавливает вид графика (свечи, бары или линия).

```
bool Mode(  
    ENUM_CHART_MODE mode           // вид графика  
)
```

### Параметры

*mode*

[in] Вид графика (свечи, бары или линия) из перечисления [ENUM\\_CHART\\_MODE](#).

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить вид.

## Foreground (метод Get)

Получает значение флага "Ценовой график на заднем плане".

```
bool Foreground() const
```

### Возвращаемое значение

Значение флага "Ценовой график на заднем плане" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается false.

## Foreground (метод Set)

Устанавливает значение флага "Ценовой график на заднем плане".

```
bool Foreground(  
    bool foreground // значение флага  
)
```

### Параметры

*foreground*

[in] Новое значение флага "Ценовой график на заднем плане".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## Shift (метод Get)

Получает значение флага «Режим отступа ценового графика от правого края».

```
bool Shift() const
```

### Возвращаемое значение

Значение флага «Режим отступа ценового графика от правого края» графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается `false`.

## Shift (метод Set)

Устанавливает значение флага «Режим отступа ценового графика от правого края».

```
bool Shift(  
    bool shift      // значение флага  
)
```

### Параметры

*shift*

[in] Новое значение флага «Режим отступа ценового графика от правого края».

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## ShiftSize (метод Get)

Получает размер отступа нулевого бара от правого края в процентах.

```
double ShiftSize() const
```

### Возвращаемое значение

Размер отступа нулевого бара от правого края в процентах графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [EMPTY\\_VALUE](#).

## ShiftSize (метод Set)

Устанавливает размер отступа нулевого бара от правого края в процентах.

```
bool ShiftSize(  
    double shift_size // значение свойства  
)
```

### Параметры

*shift\_size*

[in] Новое значение размера отступа нулевого бара от правого края в процентах.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## AutoScroll (метод Get)

Получает значение флага "Режим автоматического перехода к правому краю".

```
bool AutoScroll() const
```

### Возвращаемое значение

Значение флага "Режим автоматического перехода к правому краю" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается `false`.

## AutoScroll (метод Set)

Устанавливает значение флага "Режим автоматического перехода к правому краю".

```
bool AutoScroll(  
    bool autoscroll // значение флага  
)
```

### Параметры

*autoscroll*

[in] Новое значение флага "Режим автоматического перехода к правому краю".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## Scale (метод Get)

Получает значение свойства "Масштаб".

```
int Scale() const
```

### Возвращаемое значение

Значение свойства "Масштаб" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается 0.

## Scale (метод Set)

Устанавливает значение свойства "Масштаб".

```
bool Scale(  
    int scale        // значение свойства  
)
```

### Параметры

*scale*  
[in] Новое значение свойства "Масштаб".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## ScaleFix (метод Get)

Получает значение флага "Режим фиксированного масштаба".

```
bool ScaleFix() const
```

### Возвращаемое значение

Значение флага "Режим фиксированного масштаба" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается false.

## ScaleFix (метод Set)

Устанавливает значение флага "Режим фиксированного масштаба".

```
bool ScaleFix(  
    bool scale_fix      // значение флага  
)
```

### Параметры

*scale\_fix*  
[in] Новое значение флага "Режим фиксированного масштаба".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## ScaleFix\_11 (метод Get)

Получает значение флага "Режим масштаба 1:1".

```
bool ScaleFix_11() const
```

### Возвращаемое значение

Значение флага "Режим масштаба 1:1" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается false.

## ScaleFix\_11 (метод Set)

Устанавливает значение флага «Режим масштаба 1:1».

```
bool ScaleFix_11(  
    string scale_11          // значение флага  
)
```

### Параметры

*scale\_11*  
[in] Новое значение флага "Режим масштаба 1:1".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## FixedMax (метод Get)

Получает фиксированный максимум графика.

```
double FixedMax() const
```

### Возвращаемое значение

Фиксированный максимум графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [EMPTY\\_VALUE](#).

## FixedMax (метод Set)

Устанавливает фиксированный максимум графика.

```
bool FixedMax(  
    double max          // максимум  
)
```

### Параметры

*max*

[in] Новый фиксированный максимум графика.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить фиксированный максимум графика.

## FixedMin (метод Get)

Получает фиксированный минимум графика.

```
double FixedMin() const
```

### Возвращаемое значение

Фиксированный минимум графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [EMPTY\\_VALUE](#).

## FixedMin (метод Set)

Устанавливает фиксированный минимум графика.

```
bool FixedMin(  
    double min        // минимум  
)
```

### Параметры

*min*

[in] Новый фиксированный минимум графика.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить фиксированный минимум графика.

## ScalePPB (метод Get)

Получает значение флага "Режим указания масштаба в пунктах на бар".

```
bool ScalePPB() const
```

### Возвращаемое значение

Значение флага "Режим указания масштаба в пунктах на бар" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается `false`.

## ScalePPB (метод Set)

Устанавливает значение флага "Режим указания масштаба в пунктах на бар".

```
bool ScalePPB(  
    bool scale_ppb      // значение флага  
)
```

### Параметры

*scale\_ppb*

[in] Новое значение флага "Режим указания масштаба в пунктах на бар".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## PointsPerBar (метод Get)

Получает масштаб в пунктах на бар.

```
double PointsPerBar() const
```

### Возвращаемое значение

Масштаб в пунктах на бар графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [EMPTY\\_VALUE](#).

## PointsPerBar (метод Set)

Устанавливает масштаб в пунктах на бар.

```
bool PointsPerBar(  
    double ppb          // масштаб  
)
```

### Параметры

*ppb*

[in] Новый масштаб в пунктах на бар.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить масштаб.

## ShowOHLC (метод Get)

Получает значение флага "Отображение значений OHLC".

```
bool ShowOHLC() const
```

### Возвращаемое значение

Значение флага "Отображение значений OHLC" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается false.

## ShowOHLC (метод Set)

Устанавливает значение флага "Отображение значений OHLC".

```
bool ShowOHLC(  
    bool show        // значение флага  
)
```

### Параметры

*show*

[in] Новое значение флага "Отображение значений OHLC".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## ShowLineBid (метод Get)

Получает значение флага "Отображение значения Bid горизонтальной линией".

```
bool ShowLineBid() const
```

### Возвращаемое значение

Значение флага "Отображение значения Bid горизонтальной линией" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается `false`.

## ShowLineBid (метод Set)

Устанавливает значение флага "Отображение значения Bid горизонтальной линией".

```
bool ShowLineBid(  
    bool show        // значение флага  
)
```

### Параметры

*show*

[in] Новое значение флага "Отображение значения Bid горизонтальной линией".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## ShowLineAsk (метод Get)

Получает значение флага "Отображение значения Ask горизонтальной линией".

```
bool ShowLineAsk() const
```

### Возвращаемое значение

Значение флага "Отображение значения Ask горизонтальной линией" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается `false`.

## ShowLineAsk (метод Set)

Устанавливает значение флага "Отображение значения Ask горизонтальной линией".

```
bool ShowLineAsk(  
    bool show // значение флага  
)
```

### Параметры

*show*

[in] Новое значение флага "Отображение значения Ask горизонтальной линией".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## ShowLastLine (метод Get)

Получает значение флага "Отображение значения Last горизонтальной линией".

```
bool ShowLastLine() const
```

### Возвращаемое значение

Значение флага "Отображение значения Last горизонтальной линией" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается `false`.

## ShowLastLine (метод Set)

Устанавливает значение флага "Отображение значения Last горизонтальной линией".

```
bool ShowLastLine(  
    bool show        // значение флага  
)
```

### Параметры

*show*

[in] Новое значение флага "Отображение значения Last горизонтальной линией".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## ShowPeriodSep (метод Get)

Получает значение флага "Отображение вертикальных разделителей между соседними периодами".

```
bool ShowPeriodSep() const
```

### Возвращаемое значение

Значение флага "Отображение вертикальных разделителей между соседними периодами" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается `false`.

## ShowPeriodSep (метод Set)

Устанавливает значение флага "Отображение вертикальных разделителей между соседними периодами".

```
bool ShowPeriodSep(  
    bool show      // значение флага  
)
```

### Параметры

*show*

[in] Новое значение флага "Отображение вертикальных разделителей между соседними периодами".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## ShowGrid (метод Get)

Получает значение флага "Отображение сетки".

```
bool ShowGrid() const
```

### Возвращаемое значение

Значение флага "Отображение сетки" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается false.

## ShowGrid (метод Set)

Устанавливает значение флага "Отображение сетки".

```
bool ShowGrid(  
    bool show        // значение флага  
)
```

### Параметры

*show*  
[in] Новое значение флага "Отображение сетки".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## ShowVolumes (метод Get)

Получает значение флага "Отображение объемов".

```
bool ShowVolumes()
```

### Возвращаемое значение

Значение флага "Отображение объемов" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается false.

## ShowVolumes (метод Set)

Устанавливает значение флага "Отображение объемов".

```
bool ShowVolumes(  
    bool show        // значение флага  
)
```

### Параметры

*show*

[in] Новое значение флага "Отображение объемов".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## ShowObjectDescr (метод Get)

Получает значение флага "Отображение всплывающих описаний графических объектов".

```
bool ShowObjectDescr() const
```

### Возвращаемое значение

Значение флага "Отображение всплывающих описаний графических объектов" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается `false`.

## ShowObjectDescr (метод Get)

Устанавливает значение флага "Отображение всплывающих описаний графических объектов".

```
bool ShowObjectDescr(  
    bool show        // значение флага  
)
```

### Параметры

*show*

[in] Новое значение флага "Отображение всплывающих описаний графических объектов".

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось изменить флаг.

## ShowDateScale

Устанавливает значение флага "Отображение шкалы времени".

```
bool ShowDateScale(  
    bool show        // значение флага  
)
```

### Параметры

*show*

[in] Новое значение флага "Отображение шкалы времени".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## ShowPriceScale

Устанавливает значение флага "Отображение ценовой шкалы".

```
bool ShowPriceScale(  
    bool show      // значение флага  
)
```

### Параметры

*show*

[in] Новое значение флага "Отображение ценовой шкалы".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## ColorBackground (метод Get)

Получает цвет фона.

```
color ColorBackground() const
```

### Возвращаемое значение

Значение цвета фона графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorBackground (метод Set)

Устанавливает цвет фона.

```
bool ColorBackground(  
    color new_color // цвет  
)
```

### Параметры

*new\_color*  
[in] Новый цвет фона.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorForeground (метод Get)

Получает цвет осей, шкалы и строки OHLC.

```
color ColorForeground() const
```

### Возвращаемое значение

Цвет осей, шкалы и строки OHLC графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorForeground (метод Set)

Устанавливает цвет осей, шкалы и строки OHLC.

```
bool ColorForeground(  
    color new_color // цвет  
)
```

### Параметры

*new\_color*

[in] Новый цвет осей, шкалы и строки OHLC.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorGrid (метод Get)

Получает цвет сетки.

```
color ColorGrid() const
```

### Возвращаемое значение

Цвет сетки графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorGrid (метод Set)

Устанавливает цвет сетки.

```
bool ColorGrid(  
    color new_color // цвет  
)
```

### Параметры

*new\_color*

[in] Новый цвет сетки.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorBarUp (метод Get)

Получает цвет бара вверх, тени и окантовки тела бычьей свечи.

```
color ColorBarUp() const
```

### Возвращаемое значение

Цвет бара вверх, тени и окантовки тела бычьей свечи графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorBarUp (метод Set)

Устанавливает цвет бара вверх, тени и окантовки тела бычьей свечи.

```
bool ColorBarUp(  
    color new_color          // цвет  
)
```

### Параметры

*new\_color*

[in] Новый цвет бара вверх, тени и окантовки тела бычьей свечи.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorBarDown (метод Get)

Получает цвет бара вниз, тени и окантовки тела медвежьей свечи.

```
color ColorBarDown() const
```

### Возвращаемое значение

Цвет бара вниз, тени и окантовки тела медвежьей свечи графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorBarDown (метод Set)

Устанавливает цвет бара вниз, тени и окантовки тела медвежьей свечи.

```
bool ColorBarDown(  
    color new_color        // цвет  
)
```

### Параметры

*new\_color*

[in] Новый цвет бара вниз, тени и окантовки тела медвежьей свечи.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorCandleBull (метод Get)

Получает цвет тела бычьей свечи.

```
color ColorCandleBull() const
```

### Возвращаемое значение

Цвет тела бычьей свечи графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorCandleBull (метод Set)

Устанавливает цвет тела бычьей свечи.

```
bool ColorCandleBull(  
    color new_color // цвет  
)
```

### Параметры

*new\_color*

[in] Новый цвет тела бычьей свечи.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorCandleBear (метод Get)

Получает цвет тела медвежьей свечи.

```
color ColorCandleBear() const
```

### Возвращаемое значение

Цвет тела медвежьей свечи графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorCandleBear (метод Set)

Устанавливает цвет тела медвежьей свечи.

```
bool ColorCandleBear(  
    color new_color        // цвет  
)
```

### Параметры

*new\_color*

[in] Новый цвет тела медвежьей свечи.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorChartLine (метод Get)

Получает цвет линии графика и японских свечей "Доджи".

```
color ColorChartLine() const
```

### Возвращаемое значение

Цвет линии графика и японских свечей "Доджи" графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorChartLine (метод Set)

Устанавливает цвет линии графика и японских свечей "Доджи".

```
bool ColorChartLine(  
    color new_color        // цвет  
)
```

### Параметры

*new\_color*

[in] Новый цвет линии графика и японских свечей "Доджи".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorVolumes (метод Get)

Получает цвет объемов и уровней открытия позиций.

```
color ColorVolumes() const
```

### Возвращаемое значение

Цвет объемов и уровней открытия позиций графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorVolumes (метод Set)

Устанавливает цвет объемов и уровней открытия позиций.

```
bool ColorVolumes(  
    color new_color        // цвет  
)
```

### Параметры

*new\_color*

[in] Новый цвет объемов и уровней открытия позиций.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorLineBid (метод Get)

Получает цвет линии Bid-цены.

```
color ColorLineBid() const
```

### Возвращаемое значение

Цвет линии Bid-цены графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorLineBid (метод Set)

Устанавливает цвет линии Bid-цены.

```
bool ColorLineBid(  
    color new_color        // цвет  
)
```

### Параметры

*new\_color*  
[in] Новый цвет линии Bid-цены.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorLineAsk (метод Get)

Получает цвет линии Ask-цены.

```
color ColorLineAsk() const
```

### Возвращаемое значение

Цвет линии Ask-цены графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorLineAsk (метод Set)

Устанавливает цвет линии Ask-цены.

```
bool ColorLineAsk(  
    color new_color // цвет  
)
```

### Параметры

*new\_color*  
[in] Новый цвет линии Ask-цены.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorLineLast (метод Get)

Получает цвет линии цены последней совершенной сделки.

```
color ColorLineLast() const
```

### Возвращаемое значение

Цвет линии цены последней совершенной сделки графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorLineLast (метод Set)

Устанавливает цвет линии цены последней совершенной сделки.

```
bool ColorLineLast(  
    color new_color          // цвет  
)
```

### Параметры

*new\_color*

[in] Новый цвет линии цены последней совершенной сделки.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## ColorStopLevels (метод Get)

Получает цвет уровней стоп-ордеров (Stop Loss и Take Profit).

```
color ColorStopLevels() const
```

### Возвращаемое значение

Цвет уровней стоп-ордеров (Stop Loss и Take Profit) графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [CLR\\_NONE](#).

## ColorStopLevels (метод Set)

Устанавливает цвет уровней стоп-ордеров (Stop Loss и Take Profit).

```
bool ColorStopLevels(  
    color new_color // цвет  
)
```

### Параметры

*new\_color*

[in] Новый цвет уровней стоп-ордеров (Stop Loss и Take Profit).

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить цвет.

## VisibleBars

Получает количество баров на графике, доступных для отображения.

```
int VisibleBars() const
```

### Возвращаемое значение

Количество баров, доступных для отображения на графике, привязанном к экземпляру класса. Если нет привязанного графика, возвращается 0.

## WindowsTotal

Получает общее количество окон графика, включая подокна индикаторов.

```
int WindowsTotal() const
```

### Возвращаемое значение

Общее количество окон, включая подокна индикаторов графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается 0.

## WindowIsVisible

Получает видимость подокон.

```
bool WindowIsVisible(  
    int num // подокно  
) const
```

### Параметры

*num*

[in] Номер подокна (0-основное окно).

### Возвращаемое значение

Видимость подокон графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается false.

## WindowHandle

Получает хэндл графика (HWND).

```
int WindowHandle() const
```

### Возвращаемое значение

Хэндл (HWND) графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [INVALID\\_HANDLE](#).

## FirstVisibleBar

Получает номер первого видимого бара на графике.

```
int FirstVisibleBar() const
```

### Возвращаемое значение

Номер первого видимого бара графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается -1.

## WidthInBars

Получает ширину графика в барах.

```
int WidthInBars() const
```

### Возвращаемое значение

Ширину в барах графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается 0.

## WidthInPixels

Получает ширину графика в пикселях.

```
int WidthInPixels() const
```

### Возвращаемое значение

Ширину в пикселях графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается 0.

## HeightInPixels

Получает высоту окна в пикселях.

```
int HeightInPixels(  
    int num // подокно  
) const
```

### Параметры

*num*

[in] Номер проверяемого подокна (0-основное окно).

### Возвращаемое значение

Высоту окна в пикселях графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается 0.

## PriceMin

Получает минимум окна.

```
double PriceMin(  
    int num // подокно  
) const
```

### Параметры

*num*

[in] Номер подокна (0-основное окно).

### Возвращаемое значение

Минимум окна графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [EMPTY\\_VALUE](#).

## PriceMax

Получает максимум окна.

```
double PriceMax(  
    int num // подокно  
) const
```

### Параметры

*num*

[in] Номер подокна (0-основное окно).

### Возвращаемое значение

Максимум окна графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается [EMPTY\\_VALUE](#).

## Attach

Привязывает текущий график к экземпляру класса.

```
void Attach()
```

## Attach

Привязывает указанный график к экземпляру класса.

```
void Attach(  
    long chart           // идентификатор графика  
)
```

### Параметры

*chart*

[in] Идентификатор привязываемого графика.

## FirstChart

Привязывает первый график клиентского терминала к экземпляру класса.

```
void FirstChart()
```

## NextChart

Привязывает следующий за уже привязанным, график к экземпляру класса.

```
void NextChart()
```

## Open

Открывает график с указанными параметрами и привязывает его к экземпляру класса.

```
long Open(
    const string     symbol_name,      // инструмент
    ENUM_TIMEFRAMES timeframe        // период
)
```

### Параметры

*symbol\_name*

[in] Символ графика. [NULL](#) означает символ текущего графика (к которому прикреплен эксперт).

*timeframe*

[in] Таймфрейм графика (из перечисления [ENUM\\_TIMEFRAMES](#)). 0 означает период текущего графика.

### Возвращаемое значение

идентификатор графика

## Detach

Отвязывает график от экземпляра класса.

```
void Detach()
```

## Close

Закрывает график привязанный к экземпляру класса.

```
void Close()
```

## BringToTop

Показывает график поверх всех других.

```
bool BringToTop() const
```

### Возвращаемое значение

true в случае успеха, false в случае ошибки.

## EventObjectCreate

Устанавливает флаг отправки сообщений о [событиях](#) создания графического объекта.

```
bool EventObjectCreate(
    bool flag        // флаг
)
```

### Параметры

*flag*

[in] Новое значение флага отправки сообщений о событиях создания графических объектов.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## EventObjectDelete

Устанавливает флаг отправки сообщений о [событиях](#) удаления графических объектов.

```
bool EventObjectDelete(
    bool flag        // флаг
)
```

### Параметры

*flag*

[in] Новое значение флага отправки сообщений о событиях удаления графических объектов.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

## IndicatorAdd

Добавляет на указанное окно графика индикатор с указанным хэндлом.

```
bool IndicatorAdd(
    int    sub_win          // номер подокна
    int    handle           // хэндл индикатора
);
```

### Параметры

*sub\_win*

[in] Номер подокна графика. 0 означает главное окно графика. Если указан номер несуществующего окна, то будет создано новое окно.

*handle*

[in] Хэндл индикатора.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Смотри также

[IndicatorDelete\(\)](#), [IndicatorsTotal\(\)](#), [IndicatorName\(\)](#).

## IndicatorDelete

Удаляет с указанного окна графика индикатор с указанным именем.

```
bool IndicatorDelete(
    int         sub_win      // номер подокна
    const string name        // короткое имя индикатора
);
```

### Параметры

*sub\_win*

[in] Номер подокна графика. 0 означает главное окно графика.

*const name*

[in] Короткое имя индикатора, которое задается в свойстве [INDICATOR\\_SHORTNAME](#) функцией [IndicatorSetString\(\)](#). Получить короткое имя индикатора можно функцией [IndicatorName\(\)](#).

### Возвращаемое значение

Возвращает true в случае успешного удаления индикатора, иначе false. Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Если в указанном подокне графика существует несколько индикаторов с одинаковым коротким именем, то будет удален первый по порядку.

Если на значениях удаляемого индикатора построены другие индикаторы на этом же графике, то они также будут удалены.

Не следует путать короткое имя индикатора и имя файла, которое указывается при создании индикатора функциями [iCustom\(\)](#) и [IndicatorCreate\(\)](#). Если короткое наименование индикатора не задается явным образом, то при компиляции в нем указывается имя файла, содержащего исходный код индикатора.

Удаление индикатора с графика не означает, что расчетная часть индикатора также будет удалена из памяти терминала. Для освобождения хэндла индикатора используйте функцию [IndicatorRelease\(\)](#).

Необходимо правильно формировать короткое имя индикатора, которое с помощью функции [IndicatorSetString\(\)](#) записывается в свойство [INDICATOR\\_SHORTNAME](#). Мы рекомендуем, чтобы короткое имя содержало значения входных параметров индикатора, так как идентификация удаляемого с графика индикатора в функции [IndicatorDelete\(\)](#) производится именно по короткому имени.

### Смотри также

[IndicatorAdd\(\)](#), [IndicatorsTotal\(\)](#), [IndicatorName\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#).

## IndicatorsTotal

Возвращает количество всех индикаторов, присоединенных к указанному окну графика.

```
int IndicatorsTotal(
    long chart_id,           // идентификатор графика
    int sub_window           // номер подокна
);
```

### Параметры

*chart\_id*

[in] Идентификатор графика. 0 означает текущий график.

*sub\_window*

[in] Номер подокна графика. 0 означает главное окно графика.

### Возвращаемое значение

Количество индикаторов на указанном окне графика. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Функция предназначена для организации перебора всех индикаторов, присоединенных к данному графику. Количество всех окон графика можно получить из свойства [CHART\\_WINDOWS\\_TOTAL](#) функцией [GetInteger\(\)](#).

### Смотри также

[IndicatorAd\(\)](#), [IndicatorDelete\(\)](#), [IndicatorsTotal\(\)](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#),  
[IndicatorSetString\(\)](#).

## IndicatorName

Возвращает короткое имя индикатора по номеру в списке индикаторов на указанном окне графика.

```
string IndicatorName(
    int sub_win      // номер подокна
    int index        // индекс индикатора в списке индикаторов, добавленных к данному
);
```

### Параметры

*sub\_win*

[in] Номер подокна графика. 0 означает главное окно графика.

*index*

[in] Индекс индикатора с списке индикаторов. Нумерация индикаторов начинается с нуля, то есть самый первый индикатор в списке имеет нулевой индекс. Количество индикаторов в списке можно получить функцией [IndicatorsTotal\(\)](#).

### Возвращаемое значение

Короткое имя индикатора, которое задается в свойстве [INDICATOR\\_SHORTNAME](#) функцией [SetString\(\)](#). Чтобы получить информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

### Примечание

Не следует путать короткое имя индикатора и имя файла, которое указывается при создании индикатора функциями [iCustom\(\)](#) и [IndicatorCreate\(\)](#). Если короткое наименование индикатора не задается явным образом, то при компиляции в нем указывается имя файла с исходным кодом индикатора.

Необходимо правильно формировать короткое имя индикатора, которое с помощью функции [IndicatorSetString\(\)](#) записывается в свойство [INDICATOR\\_SHORTNAME](#). Мы рекомендуем, чтобы короткое имя содержало значения входных параметров индикатора, так как идентификация удаляемого с графика индикатора в функции [IndicatorDelete\(\)](#) производится именно по короткому имени.

### Смотри также

[IndicatorAdd\(\)](#), [IndicatorDelete](#), [IndicatorsTotal](#), [iCustom\(\)](#), [IndicatorCreate\(\)](#), [IndicatorSetString\(\)](#).

## Navigate

Сдвигает график.

```
bool Navigate(
    ENUM_CHART_POSITION position,      // позиция
    int                  shift=0        // сдвиг
)
```

### Параметры

*position*

[in] Позиция графика (из перечисления [ENUM\\_CHART\\_POSITION](#)), относительно которой будет произведено смещение.

*shift=0*

[in] Количество баров, на которое необходимо сместить график.

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось сдвинуть график.

## Symbol

Получает имя символа графика.

```
string Symbol() const
```

### Возвращаемое значение

Имя символа графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается "".

## Period

Получает значение периода графика.

```
ENUM_TIMEFRAMES Period() const
```

### Возвращаемое значение

Период графика (из перечисления [ENUM\\_TIMEFRAMES](#)), привязанного к экземпляру класса. Если нет привязанного графика, возвращается 0.

## Redraw

Перерисовывает график, привязанный к экземпляру класса.

```
void Redraw()
```

## GetInteger

Возвращает значение соответствующего свойства указанного графика. Свойство графика должно быть типа [integer](#). Существует 2 варианта функции.

1. Непосредственно возвращает значение свойства.

```
long GetInteger(
    ENUM_CHART_PROPERTY_INTEGER prop_id,           // идентификатор свойства
    int                         sub_window=0        // номер подокна
) const
```

2. Помещает значение свойства в приемную переменную, передаваемую по ссылке последним параметром, в случае успеха.

```
bool GetInteger(
    ENUM_CHART_PROPERTY_INTEGER prop_id,           // идентификатор свойства
    int                         sub_window,          // номер подокна
    long&                      value              // ссылка на переменную
) const
```

### Параметры

*prop\_id*

[in] Идентификатор свойства графика (из перечисления [ENUM\\_CHART\\_PROPERTY\\_INTEGER](#)).

*sub\_window*

[in] Номер подокна графика.

*value*

[in] Ссылка на переменную, принимающую значение запрашиваемого свойства.

### Возвращаемое значение

Значение свойства графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается -1.

Для второго варианта вызова возвращает true, если данное свойство поддерживается и значение было помещено в переменную *value*, иначе возвращает false. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

## SetInteger

Устанавливает значение integer-свойства графика.

```
bool SetInteger(
    ENUM_CHART_PROPERTY_INTEGER prop_id,      // идентификатор свойства
    long value                      // значение
)
```

### Параметры

*prop\_id*

[in] Идентификатор свойства графика (из перечисления [ENUM\\_CHART\\_PROPERTY\\_INTEGER](#)).

*value*

[in] Новое значение свойства.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить integer-свойство.

## GetDouble

Возвращает значение соответствующего свойства указанного графика. Свойство графика должно быть типа double. Существует 2 варианта функции.

- Непосредственно возвращает значение свойства.

```
double GetDouble(
    ENUM_CHART_PROPERTY_DOUBLE prop_id,           // идентификатор свойства
    int                         sub_window=0        // номер подокна
) const
```

- Помещает значение свойства в приемную переменную, передаваемую по ссылке последним параметром, в случае успеха.

```
bool GetDouble(
    ENUM_CHART_PROPERTY_DOUBLE prop_id,           // идентификатор свойства
    int                         sub_window,         // номер подокна
    double&                     value            // ссылка на переменную
) const
```

### Параметры

*prop\_id*

[in] Идентификатор свойства графика (из перечисления [ENUM\\_CHART\\_PROPERTY\\_DOUBLE](#)).

*sub\_window*

[in] Номер подокна графика.

*value*

[in] Ссылка на переменную, принимающую значение запрашиваемого свойства.

### Возвращаемое значение

Значение свойства графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается `EMPTY_VALUE`.

Для второго варианта вызова возвращает `true` - в случае удачи, `false` - если не удалось получить значение свойства. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызвать функцию [GetLastError\(\)](#).

## SetDouble

Устанавливает значение double-свойства графика.

```
bool SetDouble(
    ENUM_CHART_PROPERTY_DOUBLE prop_id,      // идентификатор свойства
    double                      value        // значение
)
```

### Параметры

*prop\_id*

[in] Идентификатор свойства графика (из перечисления [ENUM\\_CHART\\_PROPERTY\\_DOUBLE](#)).

*value*

[in] Новое значение свойства.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить double-свойство.

## GetString

Возвращает значение соответствующего свойства указанного графика. Свойство графика должно быть типа `string`. Существует 2 варианта функции.

- Непосредственно возвращает значение свойства.

```
string GetString(
    ENUM_CHART_PROPERTY_STRING prop_id          // идентификатор свойства
) const
```

- Помещает значение свойства в приемную переменную, передаваемую по ссылке последним параметром, в случае успеха.

```
bool GetString(
    ENUM_CHART_PROPERTY_STRING prop_id,        // идентификатор свойства
    string& value                           // ссылка на переменную
) const
```

### Параметры

`prop_id`

[in] Идентификатор свойства графика (из перечисления [ENUM\\_CHART\\_PROPERTY\\_STRING](#)).

`value`

[in] Ссылка на переменную, принимающую значение запрашиваемого свойства.

### Возвращаемое значение

Значение свойства графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается "".

Для второго варианта вызова возвращает `true`, если данное свойство поддерживается и значение было помещено в переменную `value`, иначе возвращает `false`. Чтобы получить дополнительную информацию об [ошибке](#), необходимо вызывать функцию [GetLastError\(\)](#).

## SetString

Устанавливает значение string-свойства графика.

```
bool SetString(
    ENUM_CHART_PROPERTY_STRING prop_id,      // идентификатор свойства
    string                      value        // значение
)
```

### Параметры

*prop\_id*

[in] Идентификатор свойства графика (из перечисления [ENUM\\_CHART\\_PROPERTY\\_STRING](#)).

*value*

[in] Новое значение свойства.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить string-свойство.

## SetSymbolPeriod

Изменяет символ и период графика, привязанного к экземпляру класса.

```
bool SetSymbolPeriod(
    const string     symbol_name,      // инструмент
    ENUM_TIMEFRAMES timeframe        // период
)
```

### Параметры

*symbol\_name*

[in] Новый символ графика. [NULL](#) означает символ текущего графика (к которому прикреплен эксперт).

*timeframe*

[in] Новый период графика (из перечисления [ENUM\\_TIMEFRAMES](#)). 0 означает период текущего графика.

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойства.

## ApplyTemplate

Применяет к графику указанный шаблон.

```
bool ApplyTemplate(
    const string filename           // шаблон
)
```

### Параметры

*filename*

[in] Имя файла, содержащего шаблон.

### Возвращаемое значение

true - в случае удачи, false - если не удалось применить шаблон.

## ScreenShot

Обеспечивает скриншот указанного графика в его текущем состоянии в формате gif.

```
bool ScreenShot(
    string      filename,           // имя файла
    int         width,             // ширина
    int         height,            // высота
    ENUM_ALIGN_MODE align_mode=ALIGN_RIGHT // тип выравнивания
) const
```

### Параметры

*filename*

[in] Имя файла скриншота.

*width*

[in] Ширина скриншота в пикселях.

*height*

[in] Высота скриншота в пикселях.

*align\_mode=ALIGN\_RIGHT*

[in] Режим вывода узкого скриншота.

### Возвращаемое значение

true в случае успеха, иначе false.

## WindowOnDropped

Получает номер подокна графика, на которое брошен мышкой данный эксперт или скрипт.

```
int WindowOnDropped() const
```

### Возвращаемое значение

Номер подокна графика, на которое брошен мышкой данный эксперт или скрипт. 0 означает главное окно графика.

## PriceOnDropped

Получает ценовую координату, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт.

```
double PriceOnDropped() const
```

### Возвращаемое значение

Ценовая координата, соответствующая точке, в которой брошен мышкой данный эксперт или скрипт.

## TimeOnDropped

Получает временную координату, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт.

```
datetime TimeOnDropped() const
```

### Возвращаемое значение

Временная координата, соответствующая точке, в которой брошен мышкой данный эксперт или скрипт.

## XOnDropped

Получает координату по оси X, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт.

```
int XOnDropped() const
```

### Возвращаемое значение

Координата по оси X, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт.

## YOnDropped

Получает координату по оси Y, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт.

```
int YOnDropped() const
```

### Возвращаемое значение

Координата по оси Y, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт.

## Save

Сохраняет данные элемента списка в файле.

```
virtual bool Save(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции [FileOpen\(...\)](#), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Load

Загружает параметры объекта из файла.

```
virtual bool Load(
    int file_handle          // хэндл файла
)
```

### Параметры

*file\_handle*

[in] хэндл ранее открытого, при помощи функции [FileOpen\(...\)](#), бинарного файла

### Возвращаемое значение

true - в случае успешного завершения, false - в случае ошибки.

## Type

Получает идентификатор типа.

```
virtual int Type() const
```

### Возвращаемое значение

Идентификатор типа (для CChart - 0x1111).

## Научные графики

Графическая библиотека, содержащая классы и глобальные функции для быстрой отрисовки пользовательских графиков на чарте.

Библиотека предоставляет удобные готовые решения для построения осей, кривых, а также методы быстрого доступа к изменению общих свойств пользовательского графика.

Графическая библиотека размещается в рабочем каталоге терминала в папке `Include\Graphics`.

Класс	Описание
<a href="#"><u>CAxis</u></a>	Класс для работы с координатными осями
<a href="#"><u>ColorGenerator</u></a>	Класс, задающий цветовую схему по умолчанию
<a href="#"><u>CCurve</u></a>	Класс для работы с кривыми
<a href="#"><u>CGraphic</u></a>	Базовый класс для создания пользовательских графиков

## GraphPlot

Функции быстрой отрисовки кривых.

**Версия для отрисовки одной кривой по координатам Y.**

```
string GraphPlot(
    const double      &y[],                      // координаты Y
    ENUM_CURVE_TYPE  type=CURVE_POINTS          // тип кривой
)
```

### Примечание

Координатами X для данной кривой будут служить индексы массива Y.

**Версия для отрисовки одной кривой по паре координат X и Y**

```
string GraphPlot(
    const double      &x[],                      // координаты X
    const double      &y[],                      // координаты Y
    ENUM_CURVE_TYPE  type=CURVE_POINTS          // тип кривой
)
```

**Версия для отрисовки двух кривых по паре координат X и Y**

```
string GraphPlot(
    const double      &x1[],                     // координаты X
    const double      &y1[],                     // координаты Y
    const double      &x2[],                     // координаты X
    const double      &y2[],                     // координаты Y
    ENUM_CURVE_TYPE  type=CURVE_POINTS          // тип кривой
)
```

**Версия для отрисовки трех кривых по паре координат X и Y**

```
string GraphPlot(
    const double      &x1[],                     // координаты X
    const double      &y1[],                     // координаты Y
    const double      &x2[],                     // координаты X
    const double      &y2[],                     // координаты Y
    const double      &x3[],                     // координаты X
    const double      &y3[],                     // координаты Y
    ENUM_CURVE_TYPE  type=CURVE_POINTS          // тип кривой
)
```

**Версия для отрисовки кривой по координатам точек CPoint2D**

```
string GraphPlot(
    const CPoint2D &points[],           // координаты кривой
    ENUM_CURVE_TYPE type=CURVE_POINTS // тип кривой
)
```

**Версия для отрисовки двух кривых по координатам точек CPoint2D**

```
string GraphPlot(
    const CPoint2D &points1[],          // координаты кривой
    const CPoint2D &points2[],          // координаты кривой
    ENUM_CURVE_TYPE type=CURVE_POINTS // тип кривой
)
```

**Версия для отрисовки трех кривых по координатам точек CPoint2D**

```
string GraphPlot(
    const CPoint2D &points1[],          // координаты кривой
    const CPoint2D &points2[],          // координаты кривой
    const CPoint2D &points3[],          // координаты кривой
    ENUM_CURVE_TYPE type=CURVE_POINTS // тип кривой
)
```

**Версия для отрисовки кривой по указателю на функцию CurveFunction**

```
string GraphPlot(
    CurveFunction function,            // указатель на функцию
    const double from,                // начальное значение аргумента
    const double to,                  // конечное значение аргумента
    const double step,                // приращение по аргументу
    ENUM_CURVE_TYPE type=CURVE_POINTS // тип кривой
)
```

**Версия для отрисовки двух кривых по указателям на функции CurveFunction**

```
string GraphPlot(
    CurveFunction function1,           // указатель на функцию
    CurveFunction function2,           // указатель на функцию
    const double from,                // начальное значение аргумента
    const double to,                  // конечное значение аргумента
    const double step,                // приращение по аргументу
    ENUM_CURVE_TYPE type=CURVE_POINTS // тип кривой
)
```

### Версия для отрисовки трех кривых по указателям на функции CurveFunction

```
string GraphPlot(
    CurveFunction    function1,           // указатель на функцию
    CurveFunction    function2,           // указатель на функцию
    CurveFunction    function3,           // указатель на функцию
    const double     from,               // начальное значение аргумента
    const double     to,                // конечное значение аргумента
    const double     step,              // приращение по аргументу
    ENUM_CURVE_TYPE type=CURVE_POINTS // тип кривой
)
```

### Параметры

*&x[]*

[in] Координаты X.

*&y[]*

[in] Координаты Y.

*&x1[]*

[in] Координаты X для первой кривой.

*&y1[]*

[in] Координаты X для второй кривой.

*&x2[]*

[in] Координаты X для третьей кривой.

*&y2[]*

[in] Координаты Y для первой кривой.

*&x3[]*

[in] Координаты X для второй кривой.

*&y3[]*

[in] Координаты Y для третьей кривой.

*&points[]*

[in] Координаты точек кривой.

*&points1[]*

[in] Координаты точек первой кривой.

*&points2[]*

[in] Координаты точек второй кривой.

*&points3[]*

[in] Координаты точек третьей кривой.

*function*

[in] Указатель на функцию CurveFunction.

*function1*

[in] Указатель на первую функцию.

*function2*

[in] Указатель на вторую функцию.

*function3*

[in] Указатель на третью функцию.

*from*

[in] Соответствует первой координате X.

*to*

[in] Соответствует последней координате X.

*step*

[in] Параметр для расчета координат X.

*type=CURVE\_POINTS*

[in] Тип кривой.

#### Возвращаемое значение

Имя графического ресурса.

## CAxis

Класс CAxis – вспомогательный класс графической библиотеки для работы с координатными осями.

### Описание

Класс CAxis обеспечивает хранение и получение различных параметров координатных осей. В классе реализована возможность динамического автомасштабирования координатной оси.

### Декларация

```
class CAxis
```

### Заголовок

```
#include <Graphics\Axis.mqh>
```

### Методы класса

Метод	Описание
<a href="#"><u>AutoScale</u></a>	Получить/установить флаг автомасштабирования
<a href="#"><u>Min</u></a>	Получить/установить минимальное значение оси
<a href="#"><u>Max</u></a>	Получить/установить максимальное значение оси
<a href="#"><u>Step</u></a>	Возвращает значение шага по оси
<a href="#"><u>Name</u></a>	Получить/установить имя оси
<a href="#"><u>Color</u></a>	Получить/установить цвет оси
<a href="#"><u>ValuesSize</u></a>	Получить/установить размер цифр на оси
<a href="#"><u>ValuesWidth</u></a>	Получить/установить максимальную отображаемую длину цифр на оси
<a href="#"><u>ValuesFormat</u></a>	Получить/установить формат цифр на оси
<a href="#"><u>ValuesDateTimeMode</u></a>	Возвращает формат преобразования даты в строку.
<a href="#"><u>ValuesFunctionFormat</u></a>	Возвращает указатель на функцию, определяющую формат вывода значений на оси.
<a href="#"><u>ValuesFunctionFormatCBData</u></a>	Возвращает указатель на объект, в котором может содержаться дополнительная информация для преобразования значений оси.

<a href="#"><u>NameSize</u></a>	Получить/установить размер шрифта для имени оси
<a href="#"><u>ZeroLever</u></a>	Получить/установить значение "нулевого рычага"
<a href="#"><u>DefaultStep</u></a>	Получить/установить начальное значение шага по оси
<a href="#"><u>MaxLabels</u></a>	Получить/установить максимальное количество цифр на оси
<a href="#"><u>MinGrace</u></a>	Получить/установить значение "допуска" для минимума оси
<a href="#"><u>MaxGrace</u></a>	Получить/установить значение "допуска" для максимума оси
<a href="#"><u>SelectAxisScale</u></a>	Выполняет автомасштабирование оси.

## AutoScale (метод Get)

Возвращает флаг, отмечающий, нужно ли выполнять автомасштабирование.

```
bool AutoScale()
```

### Возвращаемое значение

Значение флага.

### Примечание

`true` — выполнять автомасштабирование.

`false` — не выполнять автомасштабирование.

## AutoScale (метод Set)

Устанавливает флаг, отмечающий, нужно ли выполнять автомасштабирование.

```
void AutoScale(
    const bool auto          // значение флага
)
```

### Параметры

`auto`

[in]

### Примечание

`true` — выполнять автомасштабирование.

`false` — не выполнять автомасштабирование.

## Min (метод Get)

Возвращает минимальное значение оси.

```
double Min()
```

### Возвращаемое значение

Минимальное значение оси.

## Min (метод Set)

Устанавливает минимальное значение оси.

```
void Min(  
    const double min // минимальное значение  
)
```

### Параметры

*min*

[in] Минимальное значение.

## Max (метод Get)

Возвращает максимальное значение оси.

```
double Max()
```

### Возвращаемое значение

Максимальное значение оси.

## Max (метод Set)

Устанавливает максимальное значение оси графика.

```
void Max(  
    const double max           // максимальное значение  
)
```

### Параметры

*max*

[in] Максимальное значение оси графика.

## Step (метод Get)

Возвращает значение шага по оси.

```
double Step()
```

### Возвращаемое значение

Значение шага.

## Name (метод Get)

Возвращает имя оси.

```
string Name()
```

### Возвращаемое значение

Имя оси.

## Name (метод Set)

Устанавливает имя оси.

```
void Name(  
    const string name // имя оси  
)
```

### Параметры

*name*

[in] Имя оси.

## Color (метод Get)

Возвращает цвет оси.

```
color Color()
```

### Возвращаемое значение

Цвет оси.

## Color (метод Set)

Устанавливает цвет оси.

```
void Color(  
    const color clr      // цвет оси  
)
```

### Параметры

*clr*

[in] Цвет оси.

## ValuesSize (метод Get)

Возвращает размер цифр на оси.

```
int ValuesSize()
```

### Возвращаемое значение

Размер цифр, отображаемых на оси.

## ValuesSize (метод Set)

Устанавливает размер цифр на оси.

```
void ValuesSize(  
    const int size          // размер цифр на оси  
)
```

### Параметры

*size*

[in] Размер цифр на оси

## ValuesWidth (метод Get)

Возвращает максимально допустимую длину в пикселях для отображения цифр на оси.

```
int ValuesWidth()
```

### Возвращаемое значение

Длина отображения цифр на оси в пикселях.

### Примечание

Если длина в пикселях для задаваемой цифры превышает максимально допустимую длину отображения, то она будет обрезаться и оканчиваться многоточием.

## ValuesWidth (метод Set)

Устанавливает максимально допустимую длину в пикселях для отображения цифр на оси.

```
void ValuesWidth(
    const int width          // максимально допустимая длина в пикселях
)
```

### Параметры

*width*

[in] Максимально допустимая длина отображения цифр на оси.

### Примечание

Если длина в пикселях для задаваемой цифры превышает максимально допустимую длину отображения, то она будет обрезаться и оканчиваться многоточием.

## ValuesFormat (метод Get)

Возвращает формат отображения цифр на оси.

```
string ValuesFormat()
```

### Возвращаемое значение

Формат отображения цифр.

## ValuesFormat (метод Set)

Устанавливает формат отображения цифр на оси.

```
void ValuesFormat(  
    const string format // формат отображения цифр на оси  
)
```

### Параметры

*format*

[in] Формат отображения цифр на оси.

## ValuesDateTimeMode (Метод Get)

Возвращает формат преобразования даты в строку.

```
int ValuesDateTimeMode()
```

### Возвращаемое значение

Формат преобразования даты в строку.

## ValuesDateTimeMode (Метод Set)

Устанавливает формат преобразования даты в строку.

```
void ValuesDateTimeMode(  
    const int mode           // формат преобразования даты в строку  
)
```

### Параметры

*mode*

[in] Формат преобразования.

### Примечание

Более подробно о форматах преобразования даты в строку смотрите в описании функции [TimeToString\(\)](#).

## ValuesFunctionFormat (Метод Get)

Возвращает указатель на функцию, определяющую формат вывода значений на оси.

```
DoubleToStringFunction ValuesFunctionFormat()
```

### Возвращаемое значение

Указатель на функцию, определяющую формат вывода значений на оси.

## ValuesFunctionFormat (Метод Set)

Устанавливает указатель на функцию, которая определяет формат вывода значений на оси.

```
void ValuesFunctionFormat(
    DoubleToStringFunction func      // функция для конвертирования числовых значений
)
```

### Параметры

*func*

[in] Пользовательская функция для конвертирования числовых значений в строку.

### Пример:



Формат вывода значений на оси X был изменен с помощью следующего кода:

```

//+-----+
//|                               DateAxisGraphic.mq5 |
//|           Copyright 2016, MetaQuotes Software Corp. |
//|           https://www.mql5.com |
//+-----+
#include <Graphics\Graphic.mqh>
//--- array for store values
double arrX[];
double arrY[];
//+-----+
//| Custom function for create values on X-axis          |
//+-----+
string TimeFormat(double x,void *cbdata)
{
    return(ToString((datetime)arrX[ArraySize(arrX)-(int)x-1]));
}
//+-----+
void OnStart()
{
    MqlRates rates[];
    CopyRates(Symbol(),Period(),0,100,rates);
    ArraySetAsSeries(rates,true);
    int size=ArraySize(rates);
    ArrayResize(arrX,size);
    ArrayResize(arrY,size);
    for(int i=0; i<size;++i)
    {
        arrX[i]=(double)rates[i].time;
        arrY[i]=rates[i].close;
    }
//--- create graphic
    CGraphic graphic;
    if(!graphic.Create(0,"DateAxisGraphic",0,30,30,780,380))
    {
        graphic.Attach(0,"DateAxisGraphic");
    }
//--- create curve
    CCurve *curve=graphic.CurveAdd(arrY,CURVE_LINES);
//--- gets the X-axis
    CAxis *xAxis=graphic.XAxis();
//--- sets the X-axis properties
    xAxis.AutoScale(false);
    xAxis.Type(AXIS_TYPE_CUSTOM);
    xAxis.ValuesFunctionFormat(TimeFormat);
    xAxis.DefaultStep(20.0);
//--- plot
    graphic.CurvePlotAll();
    graphic.Update();
}

```

## ValuesFunctionFormatCBData (Метод Get)

Возвращает указатель на объект, в котором может содержаться дополнительная информация для преобразования значений оси.

```
void* ValuesFunctionFormatCBData()
```

### Возвращаемое значение

Указатель на объект, в котором может содержаться дополнительная информация для преобразования значений оси.

## ValuesFunctionFormatCBData (Метод Set)

Устанавливает указатель на объект класса, в котором может содержаться дополнительная информация для преобразования значений оси.

```
void ValuesFunctionFormatCBData(
    void* cbdata      // указатель на объект класса
)
```

### Параметры

*cbdata*

[in] Указатель на объект любого класса, содержащий дополнительную информацию для преобразования значений оси

## NameSize (метод Get)

Возвращает размер шрифта для имени оси.

```
int NameSize()
```

### Возвращаемое значение

Размер шрифта, которым задается имя оси.

## NameSize (метод Set)

Устанавливает размер шрифта для имени оси.

```
void NameSize(  
    const int size // размер шрифта для имени оси  
)
```

### Параметры

*size*

[in] Размер шрифта, которым задается имя оси.

## ZeroLever (метод Get)

Возвращает значение "нулевого рычага".

```
double ZeroLever()
```

### Возвращаемое значение

"Нулевой рычаг".

### Примечание

Это значение используется для определения того, когда диапазон шкалы оси должен быть расширен, чтобы включить нулевое значение.

## ZeroLever (метод Set)

Устанавливает значение "нулевого рычага".

```
void ZeroLever(  
    const double value // значение "нулевого рычага"  
)
```

### Параметры

*value*

[in] Значение "нулевого рычага".

### Примечание

Это значение используется для определения того, когда диапазон шкалы оси должен быть расширен, чтобы включить нулевое значение.

## DefaultStep (метод Get)

Возвращает начальное значение шага по оси.

```
double DefaultStep()
```

### Возвращаемое значение

Шаг по оси.

## DefaultStep (метод Set)

Устанавливает начальное значение шага по оси.

```
void DefaultStep(  
    const double value // шаг по оси  
)
```

### Параметры

*value*

[in] Начальное значение шага по оси.

## MaxLabels (метод Get)

Возвращает максимальное допустимое количество отображаемых цифр на оси.

```
double MaxLabels()
```

### Возвращаемое значение

Максимальное количество цифр на оси.

## MaxLabels (метод Set)

Устанавливает максимальное допустимое количество отображаемых цифр на оси.

```
void MaxLabels(
    const double value           // максимальное количество
)
```

### Параметры

*value*

[in] Максимально допустимое количество отображаемых на оси цифр

## MinGrace (метод Get)

Возвращает значение "допуска", применяемое к минимуму оси.

```
double MinGrace()
```

### Возвращаемое значение

Значение "допуска" для минимума оси.

### Примечание

Это значение выражается в виде части общей длины оси. Например, пусть значения оси находятся в пределах от 4.0 до 16.0, тогда ее длина будет равна 12.0. Если MinGrace равен 0.1, то 10% от длины оси (или 1.2) будет вычтено из значения минимума. В результате ось будет покрывать интервал от 2.8 до 16.0.

## MinGrace (метод Set)

Устанавливает значение "допуска", применяемое к минимуму оси.

```
void MinGrace(  
    const double value // значение "допуска"  
)
```

### Параметры

*value*

[in] Значение "допуска", применяемое к минимуму оси.

### Примечание

Это значение выражается в виде части общей длины оси. Например, пусть значения оси находятся в пределах от 4.0 до 16.0, тогда ее длина будет равна 12.0. Если MinGrace равен 0.1, то 10% от длины оси (или 1.2) будет вычтено из значения минимума. В результате ось будет покрывать интервал от 2.8 до 16.0.

## MaxGrace (метод Get)

Возвращает значение "допуска", применяемое к максимуму оси.

```
double MaxGrace()
```

### Возвращаемое значение

Значение "допуска" для максимума оси.

### Примечание

Это значение выражается в виде части общей длины оси. Например, пусть значения оси находятся в пределах от 4.0 до 16.0, тогда ее длина будет равна 12.0. Если MaxGrace равен 0.1, то 10% от длины оси (или 1.2) будет добавлено к значению максимума. В результате ось будет покрывать интервал от 4.0 до 17.2.

## MaxGrace (метод Set)

Устанавливает значение "допуска", применяемое к максимуму оси.

```
void MaxGrace(  
    const double value // значение "допуска"  
)
```

### Параметры

*value*

[in] Значение допуска, применяемое к максимуму оси.

### Примечание

Это значение выражается в виде части общей длины оси. Например, пусть значения оси находятся в пределах от 4.0 до 16.0, тогда ее длина будет равна 12.0. Если MinGrace равен 0.1, то 10% от длины оси (или 1.2) будет вычтено из значения минимума. В результате, ось будет покрывать интервал от 2.8 до 16.0.

## SelectAxisScale

Выполняет автомасштабирование оси.

```
void SelectAxisScale()
```

## CColorGenerator

Класс CColorGenerator – вспомогательный класс графической библиотеки для работы с цветовой палитрой.

### Описание

Класс CColorGenerator содержит начальную палитру цветов, используемых по умолчанию для кривых (если цвет не указан пользователем).

Если все цвета из начальной палитры уже были использованы, то производится автоматическая генерация новых цветов и перезаполнение палитры.

### Декларация

```
class CColorGenerator
```

### Заголовок

```
#include <Graphics\ColorGenerator.mqh>
```

### Методы класса

Метод	Описание
<a href="#"><u>Next</u></a>	Возвращает следующий цвет из палитры
<a href="#"><u>Reset</u></a>	Приводит генератор к первоначальному состоянию

## Next

Возвращает следующий цвет из палитры.

```
uint Next()
```

### Возвращаемое значение

Цвет.

### Примечание

Если все цвета из палитры были перебраны, то начинается автоматическая генерация новых цветов, которыми будут замещены в палитре старые.

## Reset

Приводит генератор к первоначальному состоянию.

```
void Reset()
```

## CCurve

Класс CCurve является классом для работы со свойствами кривых, создаваемых на графике.

### Описание

Класс CCurve обеспечивает установку, хранение, получение координат и различных свойств кривых при работе с классом CGraphic.

Предусмотрены три режима отрисовки кривых: точками, линиями и гистограммой. Для каждого режима отрисовки в классе реализованы отдельные параметры.

### Декларация

```
class CCurve : public CObject
```

### Заголовок

```
#include <Graphics\Curve.mqh>
```

### Иерархия наследования

[CObject](#)

CCurve

### Методы класса

Метод	Описание
<a href="#"><u>Type</u></a>	Возвращает тип кривой
<a href="#"><u>Name</u></a>	Возвращает имя кривой
<a href="#"><u>Color</u></a>	Возвращает цвет кривой
<a href="#"><u>XMax</u></a>	Возвращает максимальное значение функции по оси X
<a href="#"><u>XMin</u></a>	Возвращает минимальное значение функции по оси X
<a href="#"><u>YMax</u></a>	Возвращает максимальное значение функции по оси Y
<a href="#"><u>YMin</u></a>	Возвращает минимальное значение функции по оси Y
<a href="#"><u>Size</u></a>	Возвращает количество точек, которые определяют кривую
<a href="#"><u>PointSize</u></a>	Получить/установить линейные размеры точек, определяющих кривую
<a href="#"><u>PointsFill</u></a>	Получить/установить флаг заливки точек, определяющих кривую

<a href="#"><u>PointsColor</u></a>	Получить/установить цвет заливки точек
<a href="#"><u>GetX</u></a>	Получает в массив значения X всех точек кривой
<a href="#"><u>GetY</u></a>	Получает в массив значения Y всех точек кривой
<a href="#"><u>LineStyle</u></a>	Получить/установить стиль линии при отрисовке кривой линиями
<a href="#"><u>LinesIsSmooth</u></a>	Получить/установить флаг сглаживания при отрисовке линиями
<a href="#"><u>LinesSmoothTension</u></a>	Получить/установить параметр сглаживания кривой при отрисовке линиями
<a href="#"><u>LinesSmoothStep</u></a>	Получить/установить длину аппроксимирующих линий для сглаживания при отрисовке линиями
<a href="#"><u>LinesWidth</u></a>	Получить/установить ширину линий при отрисовке кривой линиями.
<a href="#"><u>HistogramWidth</u></a>	Получить/установить ширину столбцов при отрисовке гистограммой
<a href="#"><u>CustomPlotCBData</u></a>	Получить/установить указатель на объект, который будет использоваться при пользовательском режиме отрисовки кривой.
<a href="#"><u>CustomPlotFunction</u></a>	Получить/установить указатель на функцию, которая реализует пользовательский режим отрисовки кривой.
<a href="#"><u>PointsType</u></a>	Получить/установить флаг, указывающий на тип точек, использующихся при отрисовке кривой точками.
<a href="#"><u>StepsDimension</u></a>	Получить/установить значение, указывающее на измерение, по которому делается шаг при ступенчатой отрисовке кривой.
<a href="#"><u>TrendLineCoefficients</u></a>	Получить/установить коэффициенты трендовой линии для записи их в массив.
<a href="#"><u>TrendLineColor</u></a>	Получить/установить цвет трендовой линии для кривой.
<a href="#"><u>TrendLineVisible</u></a>	Получить/установить флаг видимости трендовой линии.
<a href="#"><u>Update</u></a>	Обновляет координаты кривой.
<a href="#"><u>Visible</u></a>	Получить/установить флаг, указывающий, будет ли видна функция на графике.

## Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Compare](#)

## Type

Возвращает тип кривой.

```
ENUM_CURVE_TYPE Type()
```

### Возвращаемое значение

Тип кривой.

## Name

Возвращает имя кривой.

```
string Name()
```

### Возвращаемое значение

Имя кривой.

## Color

Возвращает цвет кривой.

```
uint Color()
```

### Возвращаемое значение

Цвет кривой.

## XMax

Возвращает максимальное значение для функции по оси X (только действительные числа).

```
double XMax()
```

### Возвращаемое значение

Максимальное действительное число среди всех аргументов для данной функции.

## XMin

Возвращает минимальное значение для функции по оси X (только действительные значения).

```
double XMin()
```

### Возвращаемое значение

Минимальное действительное число среди всех аргументов для данной функции.

## YMax

Возвращает максимальное значение для функции по оси Y (только действительные числа).

```
double YMax()
```

### Возвращаемое значение

Максимальное значение для данной функции по оси Y (только действительные числа).

## YMin

Возвращает минимальное значение для функции по оси Y (только действительные числа).

```
double YMin()
```

### Возвращаемое значение

Минимальное значение для данной функции по оси Y (только действительные числа).

## Size

Возвращает количество точек, которые определяют данную кривую.

```
int  Size()
```

### Возвращаемое значение

Количество точек, по которым определена кривая.

## PointSize (метод Get)

Возвращает линейные размеры точек, определяющих данную кривую при ее точечной отрисовке, в пикселях.

```
int PointsSize()
```

### Возвращаемое значение

Размер определяющих кривую точек в пикселях.

## PointSize (метод Set)

Устанавливает размер точек, определяющих данную кривую при ее точечной отрисовке, в пикселях.

```
void PointsSize(  
    const int size           // размер точек в пикселях  
)
```

### Параметры

*size*

[in] Размер точек, определяющих данную кривую, в пикселях.

## PointsFill (метод Get)

Возвращает флаг, указывающий, нужно ли выполнять заливку для точек, определяющих кривую при отрисовке точками.

```
bool PointsFill ()
```

### Возвращаемое значение

Значение флага.

### Примечание

true – выполнять заливку

false – не выполнять заливку

## PointsFill (метод Set)

Устанавливает флаг, указывающий, нужно ли выполнять заливку для точек, определяющих кривую при отрисовке точками.

```
void PointsFill(
    const bool fill           // значение флага
)
```

### Параметры

*fill*

[in] Значение флага.

### Примечание

true – выполнять заливку

false – не выполнять заливку

## PointsColor (метод Get)

Возвращает цвет заливки точек.

```
uint PointsColor ()
```

### Возвращаемое значение

Цвет заливки точек, определяющих данную кривую.

## PointsColor (метод Set)

Устанавливает цвет заливки точек

```
void PointsColor(
    const uint clr      //цвет заливки точек
)
```

### Параметры

*clr*

[in] Цвет заливки точек, определяющих данную кривую.

## GetX

Получает в массив значения X всех точек кривой.

```
void GetX(  
    double& x[] // массив для записи значений X  
)
```

### Параметры

x []

[out] Массив для получения значений X всех точек кривой.

### Примечание

Каждая точка кривой задается парой значений X и Y. Эти значения не являются координатами в пикселях для отрисовки в классе [CGraphic](#).

## GetY

Получает в массив значения Y всех точек кривой.

```
void GetY(
    double& y[]          // массив для записи значений Y
)
```

### Параметры

*y* []

[out] Массив для получения значений Y всех точек кривой.

### Примечание

Каждая точка кривой задается парой значений X и Y. Эти значения не являются координатами в пикселях для отрисовки в классе [CGraphic](#).

## LineStyle (метод Get)

Возвращает стиль линии при отрисовке кривой линиями.

```
ENUM_LINE_STYLE LineStyle()
```

### Возвращаемое значение

Стиль линии.

## LineStyle (метод Set)

Устанавливает стиль линий при отрисовке кривой линиями.

```
void LineStyle (
    ENUM_LINE_STYLE style          // стиль линии
)
```

### Параметры

*style*

[in] Стиль линии.

## LinesIsSmooth (метод Get)

Возвращает флаг, определяющий, нужно ли выполнять сглаживание при отрисовке кривой линиями.

```
bool LinesIsSmooth()
```

### Возвращаемое значение

Значение флага

### Примечание

true – выполнять сглаживание

false – не выполнять сглаживание

## LinesIsSmooth (метод Set)

Устанавливает флаг, определяющий, нужно ли выполнять сглаживание при отрисовке кривой линиями.

```
void LinesIsSmooth(
    const bool smooth      // значение флага
)
```

### Параметры

*smooth*

[in] Значение флага

### Примечание

true – выполнять сглаживание

false – не выполнять сглаживание

## LinesSmoothTension (метод Get)

Возвращает параметр сглаживания кривой при отрисовке линиями.

```
double LinesSmoothTension()
```

### Возвращаемое значение

Значение параметра сглаживания

### Примечание

Значение tension находится в диапазоне (0.0; 1.0].

## LinesSmoothTension (метод Set)

Устанавливает параметр сглаживания кривой при отрисовке линиями.

```
void LinesSmoothTension(  
    const double tension           // значение параметра  
)
```

### Параметры

*tension*

[in] Значение параметра сглаживания.

### Примечание

Значение tension находится в диапазоне (0.0; 1.0].

## LinesSmoothStep (метод Get)

Возвращает длину аппроксимирующих линий для сглаживания при отрисовке линиями.

```
double LinesSmoothStep()
```

### Возвращаемое значение

Длина аппроксимирующих линий в пикселях.

## LinesSmoothStep (метод Set)

Устанавливает длину аппроксимирующих линий для сглаживания при отрисовке линиями.

```
void LinesSmoothStep(  
    const double step           // длина линий  
)
```

### Параметры

*step*

[in] Длина аппроксимирующих линий

## LinesEndStyle (Метод Set)

Возвращает флаг, указывающий на [стиль отрисовки концов линий](#) при отрисовке кривой линиями.

```
ENUM_LINE_END LinesEndStyle()
```

### Возвращаемое значение

Значение флага, указывающего на стиль концов линий при отрисовке кривой линиями.

## LinesEndStyle (Метод Get)

Устанавливает флаг, указывающий на стиль отрисовки концов линий при отрисовке кривой линиями.

```
void LinesEndStyle(  
    ENUM_LINE_END end_style           // значение флага  
)
```

### Параметры

*end\_style*

[in] Значение флага, указывающего на стиль концов линии при отрисовке кривой линиями.

## LinesWidth (Метод Get)

Возвращает ширину линий при отрисовке кривой линиями.

```
int LinesWidth()
```

### Возвращаемое значение

Ширина линий.

## LinesWidth (Метод Set)

Устанавливает ширину линий при отрисовке кривой линиями.

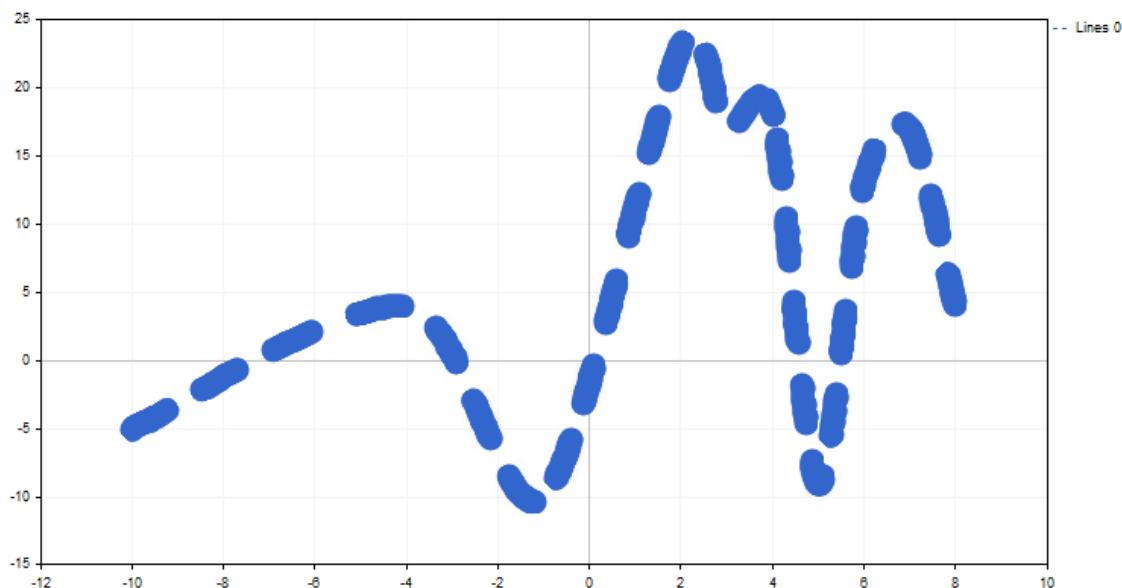
```
void LinesWidth(  
    const int width          // ширина линий  
)
```

### Параметры

*width*

[in] Ширина линий при отрисовке кривой линиями.

### Пример:



Толщина линии изменена с помощью следующего кода:

```
//+-----+
//|                                         CandleGraphic.mq5 |
//|                                         Copyright 2016, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#include <Graphics\Graphic.mqh>
//+-----+
//| Script program start function           |
//+-----+
void OnStart()
{
    double x[] = { -100,-40,-10,20,30,40,50,60,70,80,120 };
    double y[] = { -5,4,-10,23,17,18,-9,13,17,4,9 };
//--- create graphic
    CGraphic graphic;
    if(!graphic.Create(0, "ThickLineGraphic", 0, 30, 30, 780, 380))
    {
        graphic.Attach(0, "ThickLineGraphic");
    }
//--- create curve
    CCurve *curve=graphic.CurveAdd(x,y,CURVE_LINES);
//--- sets the curve properties
    curve.LinesSmooth(true);
    curve.LinesStyle(STYLE_DASH);
    curve.LinesEndStyle(LINE_END_ROUND);
    curve.LinesWidth(10);
//--- plot
    graphic.CurvePlotAll();
    graphic.Update();
}
```

## HistogramWidth (метод Get)

Возвращает ширину столбцов при отрисовке гистограммой.

```
int HistogramWidth()
```

### Возвращаемое значение

Ширина столбцов в пикселях.

## HistogramWidth (метод Set)

Устанавливает ширину столбцов при отрисовке гистограммой.

```
void HistogramWidth(  
    const int width // ширина столбцов  
)
```

### Параметры

*width*

[in] Ширина столбцов в пикселях.

## CustomPlotCBData (Метод Get)

Возвращает указатель на объект, который будет использоваться при пользовательском режиме отрисовки кривой.

```
void* CustomPlotCBData()
```

### Возвращаемое значение

Указатель на объект для пользовательского режима отрисовки кривой.

## CustomPlotCBData (Метод Set)

Устанавливает указатель на объект класса, который будет использоваться при пользовательском режиме отрисовки кривой.

```
void CustomPlotCBData(
    void* cbdata           // указатель на объект
)
```

### Параметры

*cbdata*

[in] Указатель на объект любого класса, который будет использоваться при пользовательском режиме отрисовки кривой

## CustomPlotFunction (Метод Get)

Возвращает указатель на функцию, которая реализует пользовательский режим отрисовки кривой.

```
PlotFucntion CustomPlotFunction()
```

### Возвращаемое значение

Указатель на функцию, реализующую пользовательский режим отрисовки кривой.

## CustomPlotFunction (Метод Set)

Устанавливает указатель на функцию, которая реализует пользовательский режим отрисовки кривой.

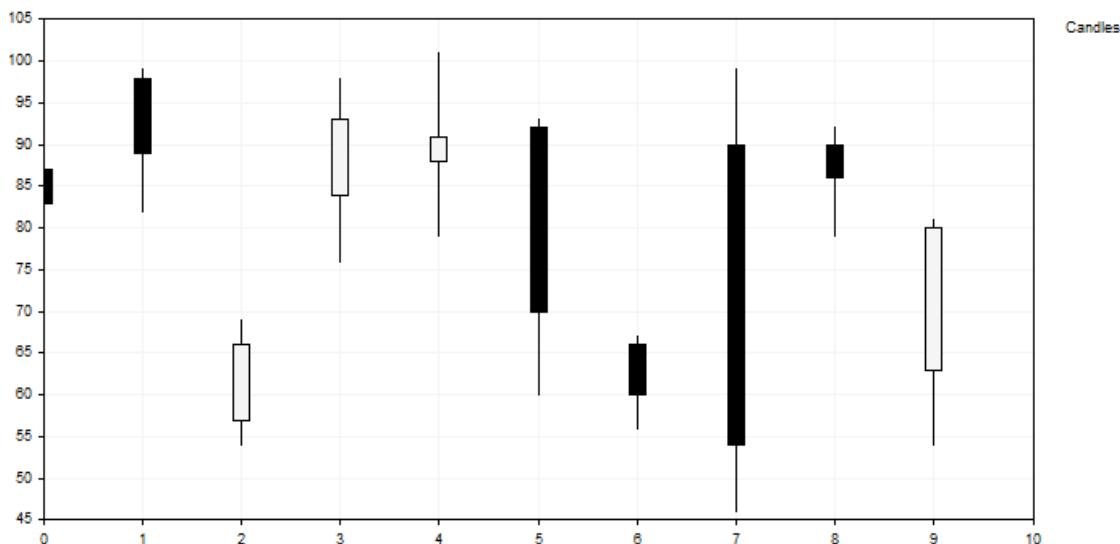
```
void CustomPlotFunction(
    PlotFucntion func      // указатель на функцию
)
```

### Параметры

*func*

[in] Указатель на функцию, реализующую пользовательский режим отрисовки кривой

### Пример:



Эта кривая из баров была построена с помощью следующего кода:

```

//+-----+
//|                                         CandleGraphic.mq5 |
//|                                         Copyright 2016, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#include <Graphics\Graphic.mqh>
//+-----+
//| Class CCandle
//| Usage: class to represent the candle
//+-----+
class CCandle: public CObject
{
private:
    double          m_open;
    double          m_close;
    double          m_high;
    double          m_low;
    uint            m_clr_inc;
    uint            m_clr_dec;
    int             m_width;

public:
    CCandle(const double open,const double close,const double high,const double low,const int width,const uint clr_inc=0x000000);
    ~CCandle(void);
    double          OpenValue(void) const { return(m_open); }
    double          CloseValue(void) const { return(m_close); }
    double          HightValue(void) const { return(m_high); }
    double          LowValue(void) const { return(m_low); }
    uint            CandleColorIncrement(void) const { return(m_clr_inc); }
    uint            CandleColorDecrement(void) const { return(m_clr_dec); }
    int             CandleWidth(void) const { return(m_width); }
};

//+-----+
//| Constructor
//+-----+
CCandle::CCandle(const double open,const double close,const double high,const double low,const int width,const uint clr_inc=0x000000,const uint clr_dec=0x000000)
{
    m_open(open),m_close(close),m_high(high),m_low(low),
    m_clr_inc(clr_inc),m_clr_dec(clr_dec),m_width(width)
}

//+-----+
//| Destructor
//+-----+
CCandle::~CCandle(void)
{
}

//+-----+
//| Custom method for plot candles
//+-----+
void PlotCandles(double &x[],double &y[],int size,CGraphic *graphic,CCanvas *canvas,void *cbdata)
{
    //--- check obj
    CArrayObj *candles=dynamic_cast<CArrayObj*>(cbdata);
    if(candles==NULL || candles.Total()!=size)
        return;
    //--- plot candles
    for(int i=0; i<size; i++)
    {
        CCandle *candle=dynamic_cast<CCandle*>(candles.At(i));
}

```

```

    if(candle==NULL)
        return;
    //--- primary calculate
    int xc=graphic.ScaleX(x[i]);
    int width_2=candle.CandleWidth()/2;
    int open=graphic.ScaleY(candle.OpenValue());
    int close=graphic.ScaleY(candle.CloseValue());
    int high=graphic.ScaleY(candle.HighthValue());
    int low=graphic.ScaleY(candle.LowValue());
    uint clr=(open<=close) ? candle.CandleColorIncrement() : candle.CandleColorDecr();
    //--- plot candle
    canvas.LineVertical(xc,high,low,0x000000);
    //--- plot candle real body
    canvas.FillRectangle(xc+width_2,open,xc-width_2,close,clr);
    canvas.Rectangle(xc+width_2,open,xc-width_2,close,0x000000);
}
}

//-----+
//| Script program start function
//+-----+
void OnStart()
{
    int count=10;
    int width=10;
    double x[];
    double y[];
    ArrayResize(x,count);
    ArrayResize(y,count);
    CArrayObj candles();
    double max=0;
    double min=0;
    //--- create values
    for(int i=0; i<count; i++)
    {
        x[i] = i;
        y[i] = i;
        //--- calculate values
        double open=MathRound(50.0+(MathRand()/32767.0)*50.0);
        double close=MathRound(50.0+(MathRand()/32767.0)*50.0);
        double high=MathRound(MathMax(open,close)+(MathRand()/32767.0)*10.0);
        double low=MathRound(MathMin(open,close)-(MathRand()/32767.0)*10.0);
        //--- find max and min
        if(i==0 || max<high)
            max=high;
        if(i==0 || min>low)
            min=low;
        //--- create candle
        CCandle *candle=new CCandle(open,close,high,low,width);
        candles.Add(candle);
    }
    //--- create graphic
    CGraphic graphic;
    if(!graphic.Create(0,"CandleGraphic",0,30,30,780,380))
    {
        graphic.Attach(0,"CandleGraphic");
    }
    //--- create curve
    CCurve *curve=graphic.CurveAdd(x,y,CURVE_CUSTOM,"Candles");
    //--- sets the curve properties
    curve.CustomPlotFunction(PlotCandles);
    curve.CustomPlotCBData(GetPointer(candles));
}

```

```
//--- sets the graphic properties
graphic.YAxis().Max((int)max);
graphic.YAxis().Min((int)min);
//--- plot
graphic.CurvePlotAll();
graphic.Update();
}
```

## PointsType (Метод Get)

Возвращает флаг, указывающий на тип точек, использующихся при отрисовке кривой точками.

```
ENUM_POINT_TYPE PointsType()
```

### Возвращаемое значение

Значение флага, указывающего на тип точек.

## PointsType (Метод Set)

Устанавливает флаг, указывающий на тип точек, использующихся при отрисовке кривой точками.

```
void PointsType(  
    ENUM_POINT_TYPE type           // значение флага  
)
```

### Параметры

*type*

[in] Значение флага, указывающего на тип точек, использующихся при отрисовке кривой точками.

## StepsDimension (Метод Get)

Возвращает значение, указывающее на измерение, по которому делается шаг при ступенчатой отрисовке кривой.

```
int StepsDimension()
```

### Возвращаемое значение

Измерение, по которому делается шаг при ступенчатой отрисовке кривой.

## StepsDimension (Метод Set)

Устанавливает значение, указывающее на измерение, по которому делается шаг при ступенчатой отрисовке кривой.

```
void StepsDimension(  
    const int dimension // измерение  
)
```

### Параметры

*dimension*

[in] Измерение (может быть либо 0, либо 1).

### Примечание

0 – x (сначала рисуется горизонтальная линия, затем вертикальная).

1 – y (сначала рисуется вертикальная линия, затем горизонтальная).

## TrendLineCoefficients (Метод Get)

Возвращает коэффициенты трендовой линии для записи их в массив.

```
double& TrendLineCoefficients()
```

### Возвращаемое значение

Коэффициенты трендовой линии.

## TrendLineCoefficients (Метод Set)

Получает коэффициенты трендовой линии для записи их в массив.

```
void TrendLineCoefficients(  
    double& coefficients[] // массив для записи коэффициентов  
)
```

### Параметры

*coefficients[]*

[out] Массив для записи коэффициентов.

## TrendLineColor (Метод Get)

Возвращает цвет трендовой линии для кривой.

```
uint TrendLineColor()
```

### Возвращаемое значение

Цвет трендовой линии.

## TrendLineColor (Метод Set)

Устанавливает цвет трендовой линии для кривой.

```
void TrendLineColor(  
    const uint clr          // цвет трендовой линии  
)
```

### Параметры

*clr*

[in] Цвет линии

## TrendLineVisible (Метод Get)

Возвращает флаг, который определяет видимость трендовой линии.

```
bool TrendLineVisible()
```

### Возвращаемое значение

Значение флага, который указывает, видна ли трендовая линия.

## TrendLineVisible (Метод Set)

Устанавливает флаг, который определяет видимость трендовой линии.

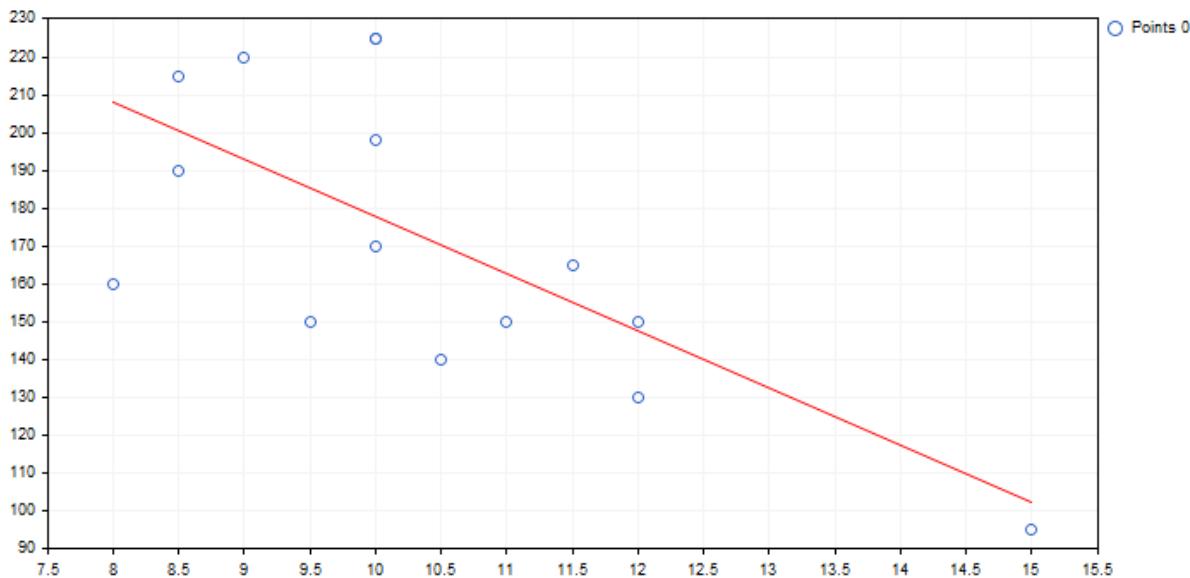
```
void TrendLineVisible(
    const bool visible           // значение флага
)
```

### Параметры

*visible*

[in] Значение флага, определяющего видимость трендовой линии.

### Пример:



Код построения вышеприведенной трендовой линии и ее отображения на графике:

```
//+-----+
//|                                              TrendLineGraphic.mq5 |
//|          Copyright 2016, MetaQuotes Software Corp. |
//|          https://www.mql5.com |
//+-----+
#include <Graphics\Graphic.mqh>
//+-----+
//| Script program start function               |
//+-----+
void OnStart()
{
    double x[]={12.0,11.5,11.0,12.0,10.5,10.0,9.0,8.5,10.0,8.5,10.0,8.0,9.5,10.0,15.0};
    double y[]={130.0,165.0,150.0,150.0,140.0,198.0,220.0,215.0,225.0,190.0,170.0,160.0
//--- create graphic
    CGraphic graphic;
    if(!graphic.Create(0,"TrendLineGraphic",0,30,30,780,380))
    {
        graphic.Attach(0,"TrendLineGraphic");
    }
//--- create curve
    CCurve *curve=graphic.CurveAdd(x,y,CURVE_POINTS);
//--- sets the curve properties
    curve.TrendLineVisible(true);
    curve.TrendLineColor(ColorToARGB(clrRed));
//--- plot
    graphic.CurvePlotAll();
    graphic.Update();
}
```

## Update

Обновляет координаты кривой.

Версия для работы по координате Y. Координатами X в данном случае будут индексы переданного массива.

```
void Update(
    const double& y[] // Координаты Y
)
```

Версия для работы по паре координат X и Y.

```
void Update(
    const double& x[], // Координаты X
    const double& y[] // Координаты Y
)
```

Версия для работы с точками CPoint2D.

```
void Update(
    const CPoint2D& points[] // Координаты кривой
)
```

Версия для работы с указателем на функцию CurveFunction.

```
void Update(
    CurveFunction function, // указатель на функцию, описывающую кривую
    const double from, // начальное значение аргумента функции
    const double to, // конечное значение аргумента функции
    const double step // приращение аргумента
)
```

### Параметры

*x[]*

[in] Координаты X.

*y[]*

[in] Координаты Y.

*points[]*

[in] Координаты кривой.

*function*

[in] Указатель на функцию, описывающую кривую

*from*

[in] Начальное значение аргумента функции

*to*

[in] Конечное значение аргумента функции

*step*

[in] Приращение аргумента

## Visible (Метод Get)

Возвращает флаг, указывающий, будет ли видна функция на графике.

```
void Visible(
    const bool visible      //
```

### Возвращаемое значение

Значение флага видимости функции на графике.

## Visible (Метод Set)

Устанавливает флаг, указывающий, будет ли видна функция на графике.

```
void Visible(
    const bool visible      // значение флага
)
```

### Параметры

*visible*

[in] Значение флага видимости функции на графике.

## CGraphic

Класс CGraphic – базовый класс для создания пользовательских графиков.

### Описание

Класс CGraphic обеспечивает многочисленные аспекты работы с пользовательскими графиками.

В классе хранятся основные элементы графика, задаются их параметры, реализуется отрисовка.

Также данный класс хранит кривые для данного графика и обеспечивает различные варианты их отображения.

### Декларация

```
class CGraphic
```

### Заголовок

```
#include <Graphics\Graphic.mqh>
```

### Методы класса

Метод	Описание
<a href="#"><u>Create</u></a>	Создает графический ресурс, привязанный к объекту чарта
<a href="#"><u>Destroy</u></a>	Удаляет с чарта график и уничтожает графический ресурс
<a href="#"><u>Update</u></a>	Отображает на экране сделанные изменения
<a href="#"><u>ChartObjectName</u></a>	Возвращает имя привязанного к графику объекта
<a href="#"><u>ResourceName</u></a>	Возвращает имя графического ресурса
<a href="#"><u>XAxis</u></a>	Возвращает указатель на ось X
<a href="#"><u>YAxis</u></a>	Возвращает указатель на ось Y
<a href="#"><u>GapSize</u></a>	Получить/установить размер отступов между элементами графика
<a href="#"><u>BackgroundColor</u></a>	Получить/установить цвет фона
<a href="#"><u>BackgroundMain</u></a>	Получить/установить текст заголовка графика
<a href="#"><u>BackgroundMainSize</u></a>	Получить/установить размер шрифта заголовка
<a href="#"><u>BackgroundMainColor</u></a>	Получить/установить цвет заголовка графика
<a href="#"><u>BackgroundSub</u></a>	Получить/установить текст подзаголовка

<a href="#"><u>BackgroundSubSize</u></a>	Получить/установить размер шрифта заголовка
<a href="#"><u>BackgroundSubColor</u></a>	Получить/установить цвет подзаголовка графика
<a href="#"><u>GridLineColor</u></a>	Получить/установить цвет линий сетки
<a href="#"><u>GridBackgroundColor</u></a>	Получить/установить цвет фона сетки
<a href="#"><u>GridCircleRadius</u></a>	Получить/установить радиус точек в узлах сетки
<a href="#"><u>GridCircleColor</u></a>	Получить/установить цвет точек в узлах сетки
<a href="#"><u>GridHasCircle</u></a>	Получить/установить флаг отрисовки точек в узлах сетки
<a href="#"><u>GridAxisLineColor</u></a>	Возвращает значение цвета реальных осей графика.
<a href="#"><u>HistoryNameWidth</u></a>	Получить/установить максимально допустимую длину для отображения имени кривой
<a href="#"><u>HistoryNameSize</u></a>	Получить/установить размер шрифта имени кривой
<a href="#"><u>HistorySymbolSize</u></a>	Получить/установить размер символов условных обозначений
<a href="#"><u>TextAdd</u></a>	Добавляет текст на график
<a href="#"><u>LineAdd</u></a>	Добавляет линию на график
<a href="#"><u>CurveAdd</u></a>	Создает и добавляет кривую на график
<a href="#"><u>CurvePlot</u></a>	Отрисовывает ранее созданную кривую по индексу
<a href="#"><u>CurvePlotAll</u></a>	Отрисовывает все ранее созданные кривые
<a href="#"><u>CurvesTotal</u></a>	Возвращает количество кривых, добавленных на график
<a href="#"><u>CurveGetByIndex</u></a>	Получает кривую по заданному индексу
<a href="#"><u>CurveGetByName</u></a>	Получает кривую по заданному имени
<a href="#"><u>CurveRemoveByIndex</u></a>	Удаляет кривую по заданному индексу.
<a href="#"><u>CurveRemoveByName</u></a>	Удаляет кривую по заданному имени.
<a href="#"><u>CurvesTotal</u></a>	Возвращает количество кривых для данного графика.
<a href="#"><u>MarksToAxisAdd</u></a>	Добавляет разметку шкалы на ось графика

<a href="#"><u>MajorMarkSize</u></a>	Получить/установить размер "засечек" шкалы на оси графика
<a href="#"><u>FontSet</u></a>	Установить параметры текущего шрифта
<a href="#"><u>FontGet</u></a>	Получить параметры текущего шрифта
<a href="#"><u>Attach</u></a>	Получить/создать графический ресурс и привязать его к экземпляру класса CGraphic
<a href="#"><u>CalculateMaxMinValues</u></a>	Выполняет расчет (перерасчет) минимальных и максимальных значений графика по обеим осям.
<a href="#"><u>Height</u></a>	Возвращает высоту графика в пикселях.
<a href="#"><u>IndentDown</u></a>	Получить/установить отступ графика от нижней границы.
<a href="#"><u>IndentLeft</u></a>	Получить/установить отступ графика от левой границы.
<a href="#"><u>IndentRight</u></a>	Получить/установить отступ графика от правой границы.
<a href="#"><u>IndentUp</u></a>	Получить/установить отступ графика от верхней границы.
<a href="#"><u>Redraw</u></a>	Перерисовывает график.
<a href="#"><u>ResetParameters</u></a>	Сбрасывает параметры, необходимые для перерисовки графика.
<a href="#"><u>ScaleX</u></a>	Масштабирует значение по оси X.
<a href="#"><u>ScaleY</u></a>	Масштабирует значение по оси Y.
<a href="#"><u>SetDefaultParameters</u></a>	Устанавливает параметры графика в значения по умолчанию.
<a href="#"><u>Width</u></a>	Возвращает ширину графика в пикселях.

## Create

Создает графический ресурс, привязанный к объекту чарта.

```
bool Create(
    const long    chart,           // идентификатор чарта
    const string   name,           // имя
    const int     subwin,          // номер подокна
    const int     x1,              // координата x1
    const int     y1,              // координата y1
    const int     x2,              // координата x2
    const int     y2               // координата y1
)
```

### Параметры

*chart*

[in] Идентификатор графика.

*name*

[in] Имя.

*subwin*

[in] Номер подокна.

*x1*

[in] Координата X1.

*y1*

[in] Координата Y1.

*x2*

[in] Координата X2.

*y2*

[in] Координата Y2.

## Destroy

Удаляет с чарта график и уничтожает графический ресурс.

```
void Destroy()
```

## Update

Отображает на экране сделанные изменения.

```
void Update(
    const bool redraw=true           // флаг
)
```

### Параметры

*redraw=true*

[in] Значение флага

## ChartObjectName

Получает имя привязанного к графику объекта.

```
string ChartObjectName()
```

### Возвращаемое значение

Имя привязанного к графику объекта.

## ResourceName

Получает имя графического ресурса.

```
string  ResourceManager()
```

### Возвращаемое значение

Имя графического ресурса.

## XAxis

Возвращает указатель на ось X.

```
CAxis *XAxis()
```

### Возвращаемое значение

Указатель на ось X.

## YAxis

Возвращает указатель на ось Y.

```
CAxis *YAxis()
```

### Возвращаемое значение

Указатель на ось Y.

## GapSize (метод Get)

Возвращает размер отступов между элементами графика.

```
int GapSize()
```

### Возвращаемое значение

Размер отступа в пикселях.

## GapSize (метод Set)

Устанавливает размер отступов между элементами графика.

```
void GapSize(  
    const int size // размер отступа  
)
```

### Параметры

*size*

[in] Размер отступа в пикселях.

## BackgroundColor (метод Get)

Возвращает цвет фона.

```
color BackgroundColor()
```

## BackgroundColor (метод Set)

Устанавливает цвет фона.

```
void BackgroundColor(  
    const color clr          // цвет фона  
)
```

### Параметры

*clr*

[in] Цвет фона.

## BackgroundMain (метод Get)

Возвращает текст заголовка графика

```
string BackgroundMain()
```

## BackgroundMain (метод Set)

Устанавливает текст заголовка графика.

```
void BackgroundMain(  
    const string main // текст заголовка  
)
```

### Параметры

*main*

[in] Текст заголовка графика

## BackgroundMainSize (метод Get)

Возвращает размер заголовка.

```
int BackgroundMainSize()
```

### Возвращаемое значение

Размер шрифта заголовка.

## BackgroundMainSize (метод Set)

Устанавливает размер заголовка.

```
void BackgroundMainSize(
    const int size      // размер заголовка
)
```

### Параметры

*size*

[in] Размер шрифта заголовка.

## BackgroundMainColor (метод Get)

Возвращает цвет заголовка.

```
color BackgroundMainColor()
```

### Возвращаемое значение

Цвет заголовка.

## BackgroundMainColor (метод Set)

Устанавливает цвет заголовка.

```
void BackgroundMainColor(  
    const color clr      // цвет заголовка  
)
```

### Параметры

*clr*

[in] Цвет заголовка.

## BackgroundSub (метод Get)

Возвращает текст подзаголовка.

```
string BackgroundSub()
```

### Возвращаемое значение

Текст подзаголовка.

## BackgroundSub (метод Set)

Устанавливает текст подзаголовка.

```
void BackgroundSub (метод Set) {
    const string sub      // текст подзаголовка
}
```

### Параметры

*sub*

[in] Текст подзаголовка.

## BackgroundSubSize (метод Get)

Возвращает размер шрифта подзаголовка

```
int BackgroundSubSize()
```

## BackgroundSubSize (метод Get)

Устанавливает размер подзаголовка

```
void BackgroundSubSize(  
    const int size          // размер шрифта подзаголовка  
)
```

### Параметры

*size*

[in] Размер шрифта подзаголовка

## BackgroundSubColor (метод Get)

Возвращает цвет подзаголовка.

```
color BackgroundSubColor()
```

## BackgroundSubColor (метод Set)

Устанавливает цвет подзаголовка.

```
void BackgroundSubColor(  
    const color clr      // цвет подзаголовка  
)
```

### Параметры

*clr*

[in] Цвет подзаголовка.

## GridLineColor (метод Get)

Возвращает цвет линий сетки

```
color GridLineColor()
```

### Возвращаемое значение

Цвет линий сетки.

## GridLineColor (метод Set)

Устанавливает цвет линий сетки.

```
void GridLineColor(  
    const color clr // цвет линий  
)
```

### Параметры

*clr*

[in] Цвет линий сетки.

## GridBackgroundColor (метод Get)

Возвращает цвет фона сетки

```
color GridBackgroundColor()
```

**Возвращаемое значение**

Цвет фона сетки

## GridBackgroundColor (метод Set)

Устанавливает цвет заднего фона сетки

```
void GridBackgroundColor(
    const color clr      // цвет фона сетки
)
```

**Параметры**

*clr*

[in] Цвет фона сетки

## GridCircleRadius (метод Get)

Возвращает радиус точек в узлах сетки.

```
int GridCircleRadius()
```

### Возвращаемое значение

Радиус точек в пикселях.

## GridCircleRadius (метод Set)

Устанавливает радиус точек в узлах сетки

```
void GridCircleRadius(  
    const int r // радиус  
)
```

### Параметры

*r*

[in] Радиус точек в пикселях.

## GridCircleColor (метод Get)

Возвращает цвет точек в узлах сетки.

```
color GridCircleColor()
```

### Возвращаемое значение

Цвет точек.

## GridCircleColor (метод Set)

Устанавливает цвет точек в узлах сетки.

```
void GridCircleColor(  
    const color clr          // цвет точек  
)
```

### Параметры

*clr*

[in] Цвет точек.

## GridHasCircle (метод Get)

Возвращает флаг, указывающий, нужно ли отрисовывать точки в узлах сетки.

```
bool GridHasCircle()
```

### Возвращаемое значение

Значение флага.

### Примечание

`true` – отрисовывать точки

`false` – не отрисовывать точки

## GridHasCircle (метод Set)

Устанавливает флаг, указывающий, нужно ли отрисовывать точки в узлах сетки.

```
void GridHasCircle(
    const bool has
)
```

### Параметры

*has*

[in] Значение флага.

### Примечание

`true` – отрисовывать точки

`false` – не отрисовывать точки

## GridAxisLineColor (Метод Get)

Возвращает значение цвета реальных осей графика.

```
uint GridAxisLineColor()
```

### Возвращаемое значение

Цвет реальных осей графика.

## GridAxisLineColor (Метод Set)

Устанавливает значение цвета реальных осей графика.

```
void GridAxisLineColor(  
    const uint clr          // цвет осей графика  
)
```

### Параметры

*clr*

[in] Цвет реальных осей графика.

## HistoryNameWidth (метод Get)

Возвращает максимально допустимую длину для отображения имени кривой.

```
int HistoryNameWidth()
```

### Возвращаемое значение

Максимальная длина в пикселях.

### Примечание

Если имя кривой превышает максимально допустимую длину, то оно будет обрезано, а в его конец будет добавлено многоточие.

## HistoryNameWidth (метод Set)

Устанавливает максимально допустимую длину для отображения имени кривой.

```
void HistoryNameWidth(  
    const int width // максимальная длина  
)
```

### Параметры

*width*

[in] Максимальная длина в пикселях.

### Примечание

Если имя кривой превышает максимально допустимую длину, то оно будет обрезано, а в его конец будет добавлено многоточие.

## HistoryNameSize (метод Get)

Возвращает размер шрифта имени кривой.

```
int HistoryNameSize()
```

### Возвращаемое значение

Размер шрифта имени кривой.

## HistoryNameSize (метод Set)

Устанавливает размер шрифта имени кривой.

```
void HistoryNameSize (метод Set) (
    const int size          // размер шрифта имени
)
```

### Параметры

*size*

[in] Размер шрифта имени.

## HistorySymbolSize (метод Get)

Возвращает размер символов условных обозначений к графику

```
int HistorySymbolSize()
```

### Возвращаемое значение

Размер символов условных обозначений

## HistorySymbolSize (метод Set)

Устанавливает размер символов условных обозначений к графику

```
void HistorySymbolSize(
    const int size          // размер символа
)
```

### Параметры

*size*

[in] Размер символов условных обозначений.

## TextAdd

Добавляет текст на график.

### Версия для работы с парой координат X и Y

```
void TextAdd(
    const int     x,           // координата X
    const int     y,           // координата Y
    const string  text,        // текст
    const uint    clr,         // цвет
    const uint    alignment=0   // выравнивание
)
```

### Версия для CPoint

```
void TextAdd(
    const CPoint  &point,      // координата точки
    const string  text,        // текст
    const uint    clr,         // цвет
    const uint    alignment=0   // выравнивание
)
```

### Параметры

*x*

[in] Координата X.

*y*

[in] Координата Y.

&*point*

[in] Координата точки.

*text*

[in] Текст.

*clr*

[in] Цвет.

*alignment=0*

[in] Выравнивание.

## LineAdd

Добавляет линию на график.

### Версия для работы с парами координат X и Y

```
void LineAdd(
    const int    x1,           // координата x1
    const int    y1,           // координата y1
    const int    x2,           // координата x2
    const int    y2,           // координата y2
    const uint   clr,          // цвет
    const uint   style         // стиль
)
```

### Версия для CPoint

```
void LineAdd2(
    const CPoint &point1,      // координата первой точки
    const CPoint &point2,      // координата второй точки
    const uint    clr,          // цвет
    const uint    style         // стиль
)
```

### Параметры

*x1*

[in] Координата X1.

*y1*

[in] Координата Y1.

*x2*

[in] Координата X2.

*y2*

[in] Координата Y2.

*&point1*

[in] Координата первой точки.

*&point2*

[in] Координата второй точки.

*clr*

[in] Цвет.

*style*

[in] Стиль.

## CurveAdd

Создаёт и добавляет новую кривую на график.

**Версия для работы по координате Y (цвет кривой задается автоматически)**

```
CCurve* CurveAdd(
    const double      &y[],           // координаты Y
    ENUM_CURVE_TYPE  type,            // тип кривой
    const string      name=NULL       // имя кривой
)
```

### Примечание

Координатами X для данной кривой будут служить индексы массива Y.

**Версия для работы по паре координат X и Y (цвет кривой задается автоматически)**

```
CCurve* CurveAdd(
    const double      &x[],           // координаты X
    const double      &y[],           // координаты Y
    ENUM_CURVE_TYPE  type,            // тип кривой
    const string      name=NULL       // имя кривой
)
```

**Версия для работы с точками CPoint2D (цвет кривой задается автоматически)**

```
CCurve* CurveAdd(
    const CPoint2D    &points[],        // координаты точек
    ENUM_CURVE_TYPE  type,             // тип кривой
    const string      name=NULL        // имя кривой
)
```

**Версия для работы с указателем на функцию CurveFunction (цвет кривой задается автоматически)**

```
CCurve* CurveAdd(
    CurveFunction     function,         // указатель на функцию
    const double      from,            // начальное значение аргумента
    const double      to,              // конечное значение аргумента
    const double      step,            // приращение по аргументу
    ENUM_CURVE_TYPE  type,             // тип кривой
    const string      name=NULL        // имя кривой
)
```

### Версия для работы по координате Y (цвет кривой задается пользователем)

```
CCurve* CurveAdd(
    const double      &y[],           // координаты Y
    const uint        clr,             // цвет кривой
    ENUM_CURVE_TYPE  type,            // тип кривой
    const string      name=NULL       // имя кривой
)
```

#### Примечание

Координатами X для данной кривой будут служить индексы массива Y.

### Версия для работы по паре координат X и Y (цвет кривой задается пользователем)

```
CCurve* CurveAdd(
    const double      &x[],           // координаты X
    const double      &y[],           // координаты Y
    const uint        clr,             // цвет кривой
    ENUM_CURVE_TYPE  type,            // тип кривой
    const string      name=NULL       // имя кривой
)
```

### Версия для работы с точками CPoint2D (цвет кривой задается пользователем)

```
CCurve* CurveAdd(
    const CPoint2D    &points[],        // координаты точек
    const uint        clr,              // цвет кривой
    ENUM_CURVE_TYPE  type,             // тип кривой
    const string      name=NULL        // имя кривой
)
```

### Версия для работы с указателем на функцию CurveFunction (цвет кривой задается пользователем)

```
CCurve* CurveAdd(
    CurveFunction     function,         // указатель на функцию
    const double      from,             // начальное значение аргумента
    const double      to,               // конечное значение аргумента
    const double      step,             // приращение по аргументу
    const uint        clr,              // цвет кривой
    ENUM_CURVE_TYPE  type,             // тип кривой
    const string      name=NULL        // имя кривой
)
```

## Параметры

*&x[]*

[in] Координата X.

*&y[]*

[in] Координата Y.

*&points[]*

[in] Координаты точек.

*function*

[in] Указатель на функцию.

*from*

[in] Начальное значение аргумента.

*to*

[in] Конечное значение аргумента.

*step*

[in] Приращение по аргументу.

*type*

[in] Тип кривой.

*name=NULL*

[in] Имя кривой.

*clr*

[in] Цвет кривой.

## Возвращаемое значение

Указатель на созданную кривую.

## CurvePlot

Отображает ранее созданную кривую с заданным индексом.

```
bool CurvePlot(
    const int index      // индекс
)
```

### Параметры

*index*

[in] Индекс кривой

### Возвращаемое значение

true в случае удачи, иначе false.

## CurvePlotAll

Отображает все ранее добавленные к графику кривые.

```
bool CurvePlotAll()
```

### Возвращаемое значение

true в случае удачи, иначе false.

## CurvesTotal

Возвращает количество кривых, добавленных к данному графику.

```
int CurvesTotal()
```

### Возвращаемое значение

Количество кривых.

## CurveGetByIndex

Получает кривую по заданному индексу.

```
CCurve* CurveGetByIndex(
    const int index // индекс кривой
)
```

### Параметры

*index*

[in] Индекс кривой.

### Возвращаемое значение

Указатель на кривую с заданным индексом.

## CurveGetByName

Получает кривую по заданному имени.

```
CCurve* CurveGetByName(
    const string name           // имя кривой
)
```

### Параметры

*name*

[in] Имя кривой.

### Возвращаемое значение

Указатель на первую найденную кривую с заданным именем.

## CurveRemoveByIndex

Удаляет кривую по заданному индексу.

```
bool CurveRemoveByIndex(
    const int index // индекс кривой
)
```

### Параметры

*index*

[in] Индекс кривой, которую требуется удалить.

### Возвращаемое значение

true – в случае удачи, иначе – false.

## CurveRemoveByName

Удаляет кривую по заданному имени.

```
bool CurveRemoveByName(
    const string name           // имя кривой
)
```

### Параметры

*name*

[in] Имя кривой, которую требуется удалить.

### Возвращаемое значение

true – в случае удачи, иначе – false.

## CurvesTotal

Возвращает количество кривых для данного графика.

```
int CurvesTotal()
```

### Возвращаемое значение

Количество кривых.

### Примечание

Считываются все кривые, принадлежащие данному графику, вне зависимости от их стиля отрисовки и видимости.

## MarksToAxisAdd

Добавляет разметку шкалы ("засечки") на указанную ось графика.

```
bool MarksToAxisAdd(
    const double      &marks[],           // координаты "засечек"
    const int         mark_size,          // размер "засечек"
    ENUM_MARK_POSITION position,         // расположение "засечек"
    const int         dimension=0        // измерение
)
```

### Параметры

*&marks[]*

[in] Координаты "засечек"

*mark\_size*

[in] Размер "засечек"

*position*

[in] Расположение "засечек"

*dimension=0*

[in] 0 – добавление на ось X,

1 – добавление на ось Y

### Возвращаемое значение

true в случае удачи, иначе false.

## MajorMarkSize (метод Get)

Возвращает размер "засечек" шкалы на координатных осях.

```
int MajorMarkSize()
```

## MajorMarkSize (метод Set)

Устанавливает размер "засечек" шкалы на координатных осях.

```
void MajorMarkSize(  
    const int size           // размер "засечек"  
)
```

### Параметры

*size*

[in] Размер "засечек" в пикселях.

## FontSet

Устанавливает параметры текущего шрифта.

```
bool FontSet(
    const string  name,           // имя
    const int     size,           // размер
    const uint    flags=0,        // флаги
    const uint    angle=0         // угол
)
```

### Параметры

*name*

[in] Имя.

*size*

[in] Размер.

*flags=0*

[in] Флаги.

*angle=0*

[in] Угол.

### Возвращаемое значение

true – в случае удачи, иначе – false.

## FontGet

Получает параметры текущего шрифта.

```
void FontGet(
    string &name,           // имя
    int    &size,           // размер
    uint   &flags,          // флаги
    uint   &angle           // угол
)
```

### Параметры

*&name*

[out] Имя.

*&size*

[out] Размер.

*&flags*

[out] Флаги.

*&angle*

[out] Угол.

## Attach

Версия для получения из объекта OBJ\_BITMAP\_LABEL графического ресурса и привязки его к экземпляру класса CGraphic:

```
bool Attach(
    const long chart_id,           // идентификатор чарта
    const string objname           // имя графического объекта
)
```

Версия для создания графического ресурса для объекта OBJ\_BITMAP\_LABEL и привязки его к экземпляру класса CGraphic:

```
bool Attach(
    const long chart_id,           // идентификатор чарта
    const string objname,          // имя графического объекта
    const int width,               // ширина картинки
    const int height                // высота картинки
)
```

### Параметры

*chart\_id*

[in] Идентификатор чарта.

*objname*

[in] Наименование (имя) графического объекта.

*width*

[in] Ширина картинки в ресурсе.

*height*

[in] Высота картинки в ресурсе.

### Возвращаемое значение

true – в случае удачи, false – если не удалось привязать объект.

## CalculateMaxMinValues

Выполняет расчет (перерасчет) минимальных и максимальных значений графика по обеим осям.

```
void CalculateMaxMinValues()
```

## Height

Возвращает высоту графика в пикселях.

```
int Height()
```

### Возвращаемое значение

Высота графика в пикселях.

## IndentDown (Метод Get)

Возвращает отступ графика от нижней границы.

```
int IndentDown()
```

### Возвращаемое значение

Размер отступа в пикселях.

## IndentDown (Метод Set)

Устанавливает отступ графика от нижней границы.

```
void IndentDown(  
    const int down // размер отступа  
)
```

### Параметры

*down*

[in] Размер отступа в пикселях.

## IndentLeft (Метод Get)

Возвращает отступ графика от левой границы.

```
int IndentLeft()
```

### Возвращаемое значение

Размер отступа в пикселях.

## IndentLeft (Метод Set)

Устанавливает отступ графика от левой границы.

```
void IndentLeft(  
    const int left // размер отступа  
)
```

### Параметры

*left*

[in] Размер отступа в пикселях.

## IndentRight (Метод Get)

Возвращает отступ графика от правой границы.

```
int IndentRight()
```

### Возвращаемое значение

Размер отступа в пикселях.

## IndentRight (Метод Set)

Устанавливает отступ графика от правой границы.

```
void IndentRight(  
    const int right // размер отступа  
)
```

### Параметры

*right*

[in] Размер отступа в пикселях.

## IndentUp (Метод Get)

Возвращает отступ графика от верхней границы.

```
int IndentUp()
```

### Возвращаемое значение

Размер отступа в пикселях.

## IndentUp (Метод Set)

Устанавливает отступ графика от верхней границы.

```
void IndentUp(  
    const int up // размер отступа  
)
```

### Параметры

*up*

[in] Значение отступа в пикселях.

## Redraw

Перерисовывает график.

```
bool Redraw(
    const bool rescale=false           // значение флага
)
```

### Параметры

*rescale=false*

[in] Флаг, указывающий, нужно ли выполнить перемасштабирование графика.

### Возвращаемое значение

true в случае удачи, иначе false.

## ResetParameters

Сбрасывает параметры, необходимые для перерисовки графика.

```
void ResetParameters()
```

## ScaleX

Масштабирует значение по оси X.

```
virtual int ScaleX(
    double x      // значение по оси X
)
```

### Параметры

x

[in] Реальное значение по оси X.

### Возвращаемое значение

Значение в пикселях.

### Примечание

Масштабирует реальное значение в пиксели для отображения на графике.

## ScaleY

Масштабирует значение по оси Y.

```
virtual int ScaleY(  
    double y      // значение по оси y  
)
```

### Параметры

*y*

[in] Реальное значение по оси y.

### Возвращаемое значение

Значение в пикселях.

### Примечание

Масштабирует реальное значение в пиксели для отображения на графике.

## SetDefaultParameters

Устанавливает параметры графика в значения по умолчанию.

```
void SetDefaultParameters()
```

## Width

Возвращает ширину графика в пикселях.

```
int Width()
```

### Возвращаемое значение

Ширина графика в пикселях.

## Технические индикаторы и таймсерии

Этот раздел содержит детали работы с классами технических индикаторов и таймсерий и, описание соответствующих компонентов стандартной библиотеки MQL5.

Использование классов технических индикаторов и таймсерий позволит сэкономить время при создании пользовательских программ (скриптов, экспертов).

Стандартная библиотека MQL5 (в части технических индикаторов и таймсерий) размещается в рабочем каталоге терминала в папке `Include\Indicators`.

Класс/группа	Описание
<a href="#"><u>Базовые классы</u></a>	Группа базовых и вспомогательных классов
<a href="#"><u>Таймсерии</u></a>	Группа классов "Таймсерии"
<a href="#"><u>Индикаторы тренда</u></a>	Группа классов "Тренд"
<a href="#"><u>Осцилляторы</u></a>	Группа классов "Осцилляторы"
<a href="#"><u>Индикаторы объема</u></a>	Группа классов "Объемы"
<a href="#"><u>Индикаторы Билла Вильямса</u></a>	Группа классов "Билл Вильямс"
<a href="#"><u>Пользовательский индикатор</u></a>	Класс "Пользовательский индикатор"

## Группа базовых и вспомогательных классов технических индикаторов и таймсерий

Этот раздел содержит детали работы с группой базовых и вспомогательных классов технических индикаторов и таймсерий и описание соответствующих компонентов стандартной библиотеки MQL5.

Класс/группа	Описание
<a href="#">CSpreadBuffer</a>	Класс буфера истории спредов
<a href="#">CTimeBuffer</a>	Класс буфера времени открытия баров
<a href="#">CTickVolumeBuffer</a>	Класс буфера тиковых объемов баров
<a href="#">CRealVolumeBuffer</a>	Класс буфера реальных объемов баров
<a href="#">CDoubleBuffer</a>	Базовый класс буфера данных типа double
<a href="#">COpenBuffer</a>	Класс буфера цен открытия баров
<a href="#">CHighBuffer</a>	Класс буфера максимальных цен баров
<a href="#">CLowBuffer</a>	Класс буфера минимальных цен баров
<a href="#">CCloseBuffer</a>	Класс буфера цен закрытия баров
<a href="#">CIndicatorBuffer</a>	Класс буфера технического индикатора
<a href="#">CSeries</a>	Базовый класс доступа к серийным данным
<a href="#">CPriceSeries</a>	Базовый класс доступа к ценовым данным
<a href="#">CIndicator</a>	Базовый класс технического индикатора
<a href="#">CIndicators</a>	Коллекция технических индикаторов и таймсерий

### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## Класс CSpreadBuffer

Класс CSpreadBuffer является классом для упрощенного доступа к историческим данным спредов.

### Описание

Класс CSpreadBuffer обеспечивает упрощенный доступ к историческим данным спредов.

### Декларация

```
class CSpreadBuffer: public CArrayInt
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject  
CArray  
CArrayInt  
CSpreadBuffer
```

### Методы класса по группам

Атрибуты	
<a href="#">Size</a>	Устанавливает размер буфера
<a href="#">Настройки</a>	
<a href="#">SetSymbolPeriod</a>	Устанавливает рабочие символ и период
<a href="#">Доступ к данным</a>	
<a href="#">At</a>	Получает элемент буфера
<a href="#">Обновление данных</a>	
<a href="#">virtual Refresh</a>	Освежает весь буфер
<a href="#">virtual RefreshCurrent</a>	Освежает только текущее значение

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayInt

[Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#),

[DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#),  
[SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

## Size

Устанавливает размер буфера.

```
void Size(  
    const int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буфера.

## SetSymbolPeriod

Устанавливает рабочие символ и период.

```
void SetSymbolPeriod(
    const string      symbol,      // символ
    const ENUM_TIMEFRAMES period   // период
)
```

### Параметры

*symbol*

[in] Новый рабочий символ.

*period*

[in] Новый рабочий период (значение перечисления [ENUM\\_TIMEFRAMES](#)).

## At

Получает элемент буфера.

```
int At(
    const int index      // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу.

## Refresh

Обновляет весь буфер.

```
virtual bool Refresh()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## RefreshCurrent

Обновляет нулевой элемент буфера.

```
virtual bool RefreshCurrent()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## Класс CTimeBuffer

Класс CTimeBuffer является классом для упрощенного доступа к историческим данным времени открытия баров.

### Описание

Класс CTimeBuffer обеспечивает упрощенный доступ к историческим данным времени открытия баров.

### Декларация

```
class CTimeBuffer: public CArrayLong
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject  
CArray  
CArrayLong  
CTimeBuffer
```

### Методы класса по группам

Атрибуты	
<a href="#">Size</a>	Устанавливает размер буфера
<b>Настройки</b>	
<a href="#">SetSymbolPeriod</a>	Устанавливает рабочие символ и период
<b>Доступ к данным</b>	
<a href="#">At</a>	Получает элемент буфера
<b>Обновление данных</b>	
<a href="#">virtual Refresh</a>	Обновляет весь буфер
<a href="#">virtual RefreshCurrent</a>	Обновляет только текущее значение

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayLong

[Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

## Size

Устанавливает размер буфера.

```
void Size(  
    const int size        // размер  
)
```

### Параметры

*size*

[in] Новый размер буфера.

## SetSymbolPeriod

Устанавливает рабочие символ и период.

```
void SetSymbolPeriod(
    const string      symbol,      // символ
    const ENUM_TIMEFRAMES period   // период
)
```

### Параметры

*symbol*

[in] Новый рабочий символ.

*period*

[in] Новый рабочий период (значение перечисления [ENUM\\_TIMEFRAMES](#)).

## At

Получает элемент буфера.

```
long At(
    const int index      // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу.

## Refresh

Обновляет весь буфер.

```
virtual bool Refresh()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## RefreshCurrent

Обновляет нулевой элемент буфера.

```
virtual bool RefreshCurrent()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## Класс CTickVolumeBuffer

Класс `CTickVolumeBuffer` является классом для упрощенного доступа к историческим данным тиковых объемов баров.

### Описание

Класс `CTickVolumeBuffer` обеспечивает упрощенный доступ к историческим данным тиковых объемов баров.

### Декларация

```
class CTickVolumeBuffer: public CArrayLong
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject  
CArray  
CArrayLong  
CTickVolumeBuffer
```

### Методы класса по группам

Атрибуты	
<a href="#">Size</a>	Устанавливает размер буфера
<b>Настройки</b>	
<a href="#">SetSymbolPeriod</a>	Устанавливает рабочие символ и период
<b>Доступ к данным</b>	
<a href="#">At</a>	Получает элемент буфера
<b>Обновление данных</b>	
<a href="#">virtual Refresh</a>	Обновляет весь буфер
<a href="#">virtual RefreshCurrent</a>	Обновляет только текущее значение

#### Методы унаследованные от `CObject`

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от `CArray`

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от `CArrayLong`

[Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

## Size

Устанавливает размер буфера.

```
void Size(  
    const int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буфера.

## SetSymbolPeriod

Устанавливает рабочие символ и период.

```
void SetSymbolPeriod(
    const string      symbol,      // символ
    const ENUM_TIMEFRAMES period   // период
)
```

### Параметры

*symbol*

[in] Новый рабочий символ.

*period*

[in] Новый рабочий период (значение перечисления [ENUM\\_TIMEFRAMES](#)).

## At

Получает элемент буфера.

```
long At(
    const int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу.

## Refresh

Обновляет весь буфер.

```
virtual bool Refresh()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## RefreshCurrent

Обновляет нулевой элемент буфера.

```
virtual bool RefreshCurrent()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## Класс CRealVolumeBuffer

Класс CRealVolumeBuffer является классом для упрощенного доступа к историческим данным реальных объемов баров.

### Описание

Класс CRealVolumeBuffer обеспечивает упрощенный доступ к историческим данным реальных объемов баров.

### Декларация

```
class CRealVolumeBuffer: public CArrayLong
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject  
CArray  
CArrayLong  
CRealVolumeBuffer
```

### Методы класса по группам

Атрибуты	
<a href="#">Size</a>	Устанавливает размер буфера
<b>Настройки</b>	
<a href="#">SetSymbolPeriod</a>	Устанавливает рабочие символ и период
<b>Доступ к данным</b>	
<a href="#">At</a>	Получает элемент буфера
<b>Обновление данных</b>	
<a href="#">virtual Refresh</a>	Обновляет весь буфер
<a href="#">virtual RefreshCurrent</a>	Обновляет только текущее значение

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayLong

[Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

## Size

Устанавливает размер буфера.

```
void Size(  
    const int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буфера.

## SetSymbolPeriod

Устанавливает рабочие символ и период.

```
void SetSymbolPeriod(
    const string      symbol,      // символ
    const ENUM_TIMEFRAMES period   // период
)
```

### Параметры

*symbol*

[in] Новый рабочий символ.

*period*

[in] Новый рабочий период (значение перечисления [ENUM\\_TIMEFRAMES](#)).

## At

Получает элемент буфера.

```
long At(
    const int index      // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу.

## Refresh

Обновляет весь буфер.

```
virtual bool Refresh()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## RefreshCurrent

Обновляет нулевой элемент буфера.

```
virtual bool RefreshCurrent()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## Класс CDoubleBuffer

Класс CDoubleBuffer является базовым классом для упрощенного доступа к данным буфера типа double.

### Описание

Класс CDoubleBuffer обеспечивает своим потомкам возможность упрощенного доступа к данным буфера типа double.

### Декларация

```
class CDoubleBuffer: public CArrayDouble
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

[CObject](#)

[CArray](#)

[CArrayDouble](#)

CDoubleBuffer

### Прямые потомки

[CCloseBuffer](#), [CHighBuffer](#), [CIndicatorBuffer](#), [CLowBuffer](#), [COpenBuffer](#)

### Методы класса по группам

Атрибуты	
<a href="#">Size</a>	Устанавливает размер буфера
<b>Настройки</b>	
<a href="#">SetSymbolPeriod</a>	Устанавливает рабочие символ и период
<b>Доступ к данным</b>	
<a href="#">At</a>	Получает элемент буфера
<b>Обновление данных</b>	
virtual <a href="#">Refresh</a>	Обновляет весь буфер
virtual <a href="#">RefreshCurrent</a>	Обновляет только текущее значение

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayDouble**

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), operator, [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

## Size

Устанавливает размер буфера.

```
void Size(  
    const int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буфера.

## SetSymbolPeriod

Устанавливает рабочие символ и период.

```
void SetSymbolPeriod(
    const string      symbol,      // символ
    const ENUM_TIMEFRAMES period   // период
)
```

### Параметры

*symbol*

[in] Новый рабочий символ.

*period*

[in] Новый рабочий период (значение перечисления [ENUM\\_TIMEFRAMES](#)).

## At

Получает элемент буфера.

```
double At(
    const int index      // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу.

## Refresh

Обновляет весь буфер.

```
virtual bool Refresh()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## RefreshCurrent

Обновляет нулевой элемент буфера.

```
virtual bool RefreshCurrent()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## Класс COpenBuffer

Класс COpenBuffer является классом для упрощенного доступа к историческим данным цен открытия баров.

### Описание

Класс COpenBuffer обеспечивает упрощенный доступ к историческим данным цен открытия баров.

### Декларация

```
class COpenBuffer: public CDoubleBuffer
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayDouble
      CDoubleBuffer
        COpenBuffer
```

### Методы класса по группам

Обновление данных	
virtual <a href="#">Refresh</a>	Обновляет весь буфер
virtual <a href="#">RefreshCurrent</a>	Обновляет только текущее значение

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), [operator](#), [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

#### Методы унаследованные от CDoubleBuffer

[Size](#), [At](#), [SetSymbolPeriod](#)

## Refresh

Обновляет весь буфер.

```
virtual bool Refresh()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## RefreshCurrent

Обновляет нулевой элемент буфера.

```
virtual bool RefreshCurrent()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## Класс CHighBuffer

Класс CHighBuffer является классом для упрощенного доступа к историческим данным максимальных цен баров.

### Описание

Класс CHighBuffer обеспечивает упрощенный доступ к историческим данным максимальных цен баров.

### Декларация

```
class CHighBuffer: public CDoubleBuffer
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayDouble
      CDoubleBuffer
        CHighBuffer
```

### Методы класса по группам

Обновление данных	
virtual <a href="#">Refresh</a>	Обновляет весь буфер
virtual <a href="#">RefreshCurrent</a>	Обновляет только текущее значение

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), [operator](#), [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

#### Методы унаследованные от CDoubleBuffer

[Size](#), [At](#), [SetSymbolPeriod](#)

## Refresh

Обновляет весь буфер.

```
virtual bool Refresh()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## RefreshCurrent

Обновляет нулевой элемент буфера.

```
virtual bool RefreshCurrent()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## Класс CLowBuffer

Класс CLowBuffer является классом для упрощенного доступа к историческим данным минимальных цен баров.

### Описание

Класс CLowBuffer обеспечивает упрощенный доступ к историческим данным минимальных цен баров.

### Декларация

```
class CLowBuffer: public CDoubleBuffer
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayDouble
      CDoubleBuffer
        CLowBuffer
```

### Методы класса по группам

Обновление данных	
virtual <a href="#">Refresh</a>	Обновляет весь буфер
virtual <a href="#">RefreshCurrent</a>	Обновляет только текущее значение

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), [operator](#), [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

#### Методы унаследованные от CDoubleBuffer

[Size](#), [At](#), [SetSymbolPeriod](#)

## Refresh

Обновляет весь буфер.

```
virtual bool Refresh()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## RefreshCurrent

Обновляет нулевой элемент буфера.

```
virtual bool RefreshCurrent()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## Класс CCloseBuffer

Класс CCloseBuffer является классом для упрощенного доступа к историческим данным цен закрытия баров.

### Описание

Класс CCloseBuffer обеспечивает упрощенный доступ к историческим данным цен закрытия баров.

### Декларация

```
class CCloseBuffer: public CDoubleBuffer
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayDouble
      CDoubleBuffer
        CCloseBuffer
```

### Методы класса по группам

Обновление данных	
virtual <a href="#">Refresh</a>	Обновляет весь буфер
virtual <a href="#">RefreshCurrent</a>	Обновляет только текущее значение

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), [operator](#), [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

#### Методы унаследованные от CDoubleBuffer

[Size](#), [At](#), [SetSymbolPeriod](#)

## Refresh

Обновляет весь буфер.

```
virtual bool Refresh()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## RefreshCurrent

Обновляет нулевой элемент буфера.

```
virtual bool RefreshCurrent()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось обновить буфер.

## Класс CIndicatorBuffer

Класс CIndicatorBuffer является классом для упрощенного доступа к данным буфера технического индикатора.

### Описание

Класс CIndicatorBuffer обеспечивает упрощенный доступ к данным буфера технического индикатора.

### Декларация

```
class CIndicatorBuffer: public CDoubleBuffer
```

### Заголовок

```
#include <Indicators\Indicator.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayDouble
      CDoubleBuffer
        CIndicatorBuffer
```

### Методы класса по группам

Атрибуты	
<a href="#">Offset</a>	Получает/устанавливает смещение буфера
<a href="#">Name</a>	Получает/устанавливает наименование буфера
<b>Доступ к данным</b>	
<a href="#">At</a>	Получает элемент буфера
<b>Обновление данных</b>	
<a href="#">Refresh</a>	Обновляет весь буфер
<a href="#">RefreshCurrent</a>	Обновляет только текущее значение

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayDouble

[Delta](#), [Type](#), [Save](#), [Load](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [InsertArray](#), [AssignArray](#), [AssignArray](#), [At](#), [operator](#), [Minimum](#), [Maximum](#), [Update](#), [Shift](#), [Delete](#), [DeleteRange](#), [CompareArray](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#), [SearchLinear](#)

Методы унаследованные от CDoubleBuffer

[Size](#), [At](#), [SetSymbolPeriod](#)

## Offset

Получает смещение буфера.

```
int Offset() const
```

### Возвращаемое значение

Смещение буфера.

## Offset

Устанавливает смещение буфера.

```
void Offset(  
    const int offset      // смещение  
)
```

### Параметры

*offset*

[in] Новое смещение буфера.

## Name

Получает наименование буфера.

```
string Name() const
```

### Возвращаемое значение

Наименование буфера.

## Name

Устанавливает наименование буфера.

```
void Name(  
    const string name // наименование  
)
```

### Параметры

*name*

[in] Новое наименование буфера.

## At

Получает элемент буфера.

```
double At(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента в буфере.

### Возвращаемое значение

Элемент буфера по указанному индексу.

## Refresh

Обновляет весь буфер.

```
bool Refresh(
    const int handle,           // хэндл индикатора
    const int num                // номер буфера
)
```

### Параметры

*handle*

[in] Хэндл индикатора.

*num*

[in] Номер индикаторного буфера.

### Возвращаемое значение

true - в случае удачи, false - если не удалось обновить буфер.

## RefreshCurrent

Обновляет нулевой элемент буфера.

```
bool RefreshCurrent(
    const int handle,           // хэндл индикатора
    const int num                // номер буфера
)
```

### Параметры

*handle*

[in] Хэндл индикатора.

*num*

[in] Номер индикаторного буфера.

### Возвращаемое значение

true - в случае удачи, false - если не удалось обновить буфер.

## Класс CSeries

Класс CSeries является базовым классом для классов доступа к серийным данным стандартной библиотеки MQL5.

### Описание

Класс CSeries обеспечивает всем своим потомкам (классам таймсерий и индикаторов) возможность упрощенного доступа к общим функциям API MQL5 в части работы с серийными данными.

### Декларация

```
class CSeries: public CArrayObj
```

### Заголовок

```
#include <Indicators\Series.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
      CSeries
```

### Прямые потомки

[CIndicator](#), [CiRealVolume](#), [CiSpread](#), [CiTickVolume](#), [CiTime](#), [CPriceSeries](#)

### Методы класса по группам

Атрибуты	
<a href="#">Name</a>	Получает имя таймсерии или индикатора
<a href="#">BuffersTotal</a>	Получает количество буферов таймсерии или индикатора
<a href="#">Timeframe</a>	Получает флаг таймфрейма таймсерии или индикатора
<a href="#">Symbol</a>	Получает имя рабочего инструмента таймсерии или индикатора
<a href="#">Period</a>	Получает рабочий таймфрейм таймсерии или индикатора
<a href="#">RefreshCurrent</a>	Устанавливает/сбрасывает флаг обновления текущих данных
<b>Доступ к данным</b>	

<code>virtual BufferResize</code>	Устанавливает размеры буферов таймсерии или индикатора
<b>Обновление данных</b>	
<code>virtual Refresh</code>	Обновляет данные таймсерии или индикатора
<code>PeriodDescription</code>	Преобразует <code>ENUM_TIMEFRAMES</code> в строку

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

## Name

Получает имя таймсерии или индикатора.

```
string Name() const
```

### Возвращаемое значение

Имя таймсерии или индикатора.

## BuffersTotal

Получает количество буферов таймсерии или индикатора.

```
int BuffersTotal() const
```

### Возвращаемое значение

Количество буферов таймсерии или индикатора.

### Примечание

Количество буферов таймсерии всегда 1.

## Timeframe

Получает флаг таймфрейма таймсерии или индикатора.

```
int Timeframe() const
```

### Возвращаемое значение

Флаг таймфрейма таймсерии или индикатора.

### Примечание

Формируется как флаги видимости объектов.

## Symbol

Получает имя рабочего инструмента таймсерии или индикатора.

```
string Symbol() const
```

### Возвращаемое значение

Имя рабочего инструмента (символа) таймсерии или индикатора.

## Period

Получает рабочий таймфрейм таймсерии или индикатора.

```
ENUM_TIMEFRAMES Period() const
```

### Возвращаемое значение

Рабочий таймфрейм таймсерии или индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

## RefreshCurrent

Устанавливает флаг необходимости постоянного обновления текущих значений таймсерии или индикатора.

```
string RefreshCurrent(
    const bool flag          // значение
)
```

### Параметры

*flag*

[in] Новое значение флага.

### Возвращаемое значение

Нет.

## BufferSize

Получает количество доступных данных в буфере таймсерии или индикатора.

```
int BufferSize() const
```

### Возвращаемое значение

Количество доступных данных в буфере таймсерии или индикатора.

## BufferResize

Устанавливает размеры буферов таймсерии или индикатора.

```
virtual bool BufferResize(  
    const int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буферов.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Все буфера индикатора или таймсерии будут иметь одинаковый размер.

## Refresh

Обновляет данные таймсерии или индикатора.

```
virtual void Refresh(
    const int   flags        // флаги
)
```

### Параметры

*flags*

[in] Флаги обновления таймфреймов.

## PeriodDescription

Преобразует в строку значение перечисления [ENUM\\_TIMEFRAMES](#).

```
string PeriodDescription(
    const int val=0           // значение
)
```

### Параметры

*val=0*

[in] Значение для преобразования.

### Возвращаемое значение

Строка, соответствующая значению перечисления [ENUM\\_TIMEFRAMES](#).

### Примечание

Если значение не указано или равно нулю, то в строку преобразуется рабочий таймфрейм таймсерии или индикатора.

## Класс CPriceSeries

Класс CPriceSeries является базовым классом для классов доступа к ценовым данным.

### Описание

Класс CPriceSeries обеспечивает всем своим потомкам возможность упрощенного доступа к общим функциям API MQL5 в части работы с ценовыми данными.

### Декларация

```
class CPriceSeries: public CSeries
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
      CSeries
        CPriceSeries
```

### Прямые потомки

[CiClose](#), [CiHigh](#), [CiLow](#), [CiOpen](#)

### Методы класса по группам

Создание	
virtual <a href="#">BufferResize</a>	Устанавливает размеры буфера серии
Доступ к данным	
virtual <a href="#">GetData</a>	Получает указанный элемент буфера серии
Обновление данных	
virtual <a href="#">Refresh</a>	Обновляет данные серии
Поиск экстремумов	
virtual <a href="#">MinIndex</a>	Получает индекс минимального значения в указанном диапазоне
virtual <a href="#">MinValue</a>	Получает минимальное значение в указанном диапазоне
virtual <a href="#">MaxIndex</a>	Получает индекс максимального значения в указанном диапазоне

virtual <a href="#">MaxValue</a>	Получает максимальное значение в указанном диапазоне
----------------------------------	--

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

## BufferResize

Устанавливает размеры буфера серии.

```
virtual bool BufferResize(  
    const int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буфера.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

## GetData

Получает указанный элемент буфера серии.

```
double GetData(
    const int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера серии, либо [EMPTY\\_VALUE](#).

## Refresh

Обновляет данные серии.

```
virtual void Refresh(  
    const int   flags=OBJ_ALL_PERIODS           // флаги  
)
```

### Параметры

*flags=OBJ\_ALL\_PERIODS*

[in] Флаги таймфреймов.

## MinIndex

Получает индекс минимального значения в указанном интервале.

```
virtual int MinIndex(
    const int start,      // размер
    constint count        // количество
) const
```

### Параметры

*start*

[in] Стартовый индекс интервала поиска.

*count*

[in] Размер интервала поиска (количество элементов).

### Возвращаемое значение

Индекс минимального значения буфера серии в указанном интервале, либо -1.

## MinValue

Получает минимальное значение в указанном интервале.

```
virtual double MinValue(  
    const int    start,      // размер  
    const int    count,      // количество  
    int&        index       // ссылка  
) const
```

### Параметры

*start*

[in] Стартовый индекс интервала поиска.

*count*

[in] Размер интервала поиска (количество элементов).

*index*

[out] Ссылка на переменную для размещения значения индекса найденного элемента.

### Возвращаемое значение

Минимальное значение буфера серии в указанном интервале, либо [EMPTY\\_VALUE](#).

### Примечание

Индекс найденного элемента помещается по ссылке *index*.

## MaxIndex

Получает индекс максимального значения в указанном интервале.

```
virtual int MaxIndex(
    const int start,           // индекс
    const int count            // количество
) const
```

### Параметры

*start*

[in] Стартовый индекс интервала поиска.

*count*

[in] Размер интервала поиска (количество элементов).

### Возвращаемое значение

Индекс максимального значения буфера серии в указанном интервале, либо -1.

## MaxValue

Получает максимальное значение в указанном интервале.

```
virtual double MaxValue(  
    const int    start,      // размер  
    const int    count,      // количество  
    int&        index       // ссылка  
) const
```

### Параметры

*start*

[in] Стартовый индекс интервала поиска.

*count*

[in] Размер интервала поиска (количество элементов).

*index*

[out] Ссылка на переменную для размещения значения индекса найденного элемента.

### Возвращаемое значение

Максимальное значение буфера серии в указанном интервале, либо [EMPTY\\_VALUE](#).

### Примечание

Индекс найденного элемента помещается по ссылке *index*.

## Класс CIndicator

Класс CIndicator является базовым классом для классов технических индикаторов стандартной библиотеки MQL5.

### Описание

Класс CIndicator обеспечивает всем своим потомкам возможность упрощенного доступа к общим функциям API MQL5 в части работы с техническими индикаторами.

### Декларация

```
class CIndicator: public CSeries
```

### Заголовок

```
#include <Indicators\Indicator.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
```

### Прямые потомки

[CiAC](#), [CiAD](#), [CiADX](#), [CiADXWilder](#), [CiAlligator](#), [CiAMA](#), [CiAO](#), [CiATR](#), [CiBands](#), [CiBearsPower](#), [CiBullsPower](#), [CiBWMFI](#), [CiCCI](#), [CiChaikin](#), [CiCustom](#), [CiDEMA](#), [CiDeMarker](#), [CiEnvelopes](#), [CiForce](#), [CiFractals](#), [CiFrAMA](#), [CiGator](#), [Cilchimoku](#), [CiMA](#), [CiMACD](#), [CiMFI](#), [CiMomentum](#), [CiOBV](#), [CiOsMA](#), [CiRSI](#), [CiRVI](#), [CiSAR](#), [CiStdDev](#), [CiStochastic](#), [CiTEMA](#), [CiTriX](#), [CiVIDyA](#), [CiVolumes](#), [CiWPR](#)

### Методы класса по группам

Атрибуты	
<a href="#">Handle</a>	Получает хэндл индикатора
<a href="#">Status</a>	Получает статус создания индикатора
<a href="#">FullRelease</a>	Устанавливает флаг освобождения хенда
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<a href="#">BufferResize</a>	Устанавливает размеры буферов индикатора
<b>Доступ к данным</b>	
<a href="#">GetData</a>	Копирует данные из буфера индикатора
<b>Обновление данных</b>	

<a href="#">Refresh</a>	Обновляет данные индикатора
<a href="#">Поиск экстремумов</a>	
<a href="#">Minimum</a>	Получает индекс минимального значения в указанном диапазоне
<a href="#">MinValue</a>	Получает минимальное значение в указанном диапазоне
<a href="#">Maximum</a>	Получает индекс максимального значения в указанном диапазоне
<a href="#">MaxValue</a>	Получает максимальное значение в указанном диапазоне
<a href="#">Преобразование перечислений</a>	
<a href="#">MethodDescription</a>	Преобразует <a href="#">ENUM_MA_METHOD</a> в строку
<a href="#">PriceDescription</a>	Преобразует <a href="#">ENUM_APPLIED_PRICE</a> в строку
<a href="#">VolumeDescription</a>	Преобразует <a href="#">ENUM_APPLIED_VOLUME</a> в строку
<a href="#">Работа с графиком</a>	
<a href="#">AddToChart</a>	Добавляет индикатор на график
<a href="#">DeleteFromChart</a>	Удаляет индикатор с графика

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)**Методы унаследованные от CArrayObj**[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)**Методы унаследованные от CSeries**[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

## Handle

Получает хэндл индикатора.

```
int Handle() const
```

### Возвращаемое значение

Хэндл индикатора.

## Status

Получает статус создания индикатора.

```
string Status() const
```

### Возвращаемое значение

Статус создания индикатора.

## FullRelease

Устанавливает флаг освобождения хендла.

```
void FullRelease(
    const bool flag=true           // флаг
)
```

### Параметры

*flag*

[in] Новое значение флага освобождения хендла.

## Create

Создает индикатор указанного типа с указанными параметрами.

```
bool Create(
    const string      symbol,           // имя символа
    const ENUM_TIMEFRAMES period,       // период
    const ENUM_INDICATOR type,          // тип
    const int          num_params,       // количество параметров
    const MqlParam&   params[]         // массив параметров
)
```

### Параметры

*symbol*

[in] Имя символа индикатора.

*period*

[in] Таймфрейм индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*type*

[in] Тип индикатора (значение перечисления [ENUM\\_INDICATOR](#)).

*num\_params*

[in] Количество параметров для создания индикатора.

*params*

[in] Ссылка на массив параметров для создания индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## BufferResize

Устанавливает размеры буферов индикатора.

```
virtual bool BufferResize(  
    const int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буферов.

### Возвращаемое значение

Возвращает true в случае успеха, иначе false.

### Примечание

Все буферы индикатора будут иметь одинаковый размер.

## BarsCalculated

Возвращает количество рассчитанных данных для индикатора.

```
int BarsCalculated() const;
```

### Возвращаемое значение

Возвращает количество рассчитанных данных в индикаторном буфере или -1 в случае ошибки (данные еще не рассчитаны).

## GetData

Получает указанный элемент указанного буфера индикатора. Для работы с актуальными данными перед использованием необходимо вызвать [Refresh\(\)](#).

```
double GetData(
    const int buffer_num,      // номер буфера
    const int index            // индекс
) const
```

### Параметры

*buffer\_num*

[in] Номер буфера индикатора.

*index*

[in] Индекс элемента буфера индикатора.

### Возвращаемое значение

значение - в случае удачи, [EMPTY\\_VALUE](#) - если не удалось получить данные.

## GetData

Получает данные из буфера индикатора по стартовой позиции и количеству.

```
int GetData(
    const int      start_pos,      // позиция
    const int      count,          // количество
    const int      buffer_num,     // номер буфера
    double&       buffer[]        // массив
) const
```

### Параметры

*start\_pos*

[in] Стартовая позиция буфера индикатора.

*count*

[in] Количество элементов буфера индикатора.

*buffer\_num*

[in] Номер буфера индикатора.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

Количество полученных значений индикатора из указанного буфера в случае удачи, иначе -1.

## GetData

Получает данные из буфера индикатора по начальному времени и количеству.

```
int GetData(
    const datetime start_time,      // начальное время
    const int count,                // количество
    const int buffer_num,           // номер буфера
    double& buffer[]               // массив
) const
```

**Параметры***start\_time***[in]** Время начального элемента буфера индикатора.*count***[in]** Количество элементов буфера индикатора.*buffer\_num***[in]** Номер буфера индикатора.*buffer***[in]** Ссылка на массив для размещения данных.**Возвращаемое значение**

Количество полученных значений индикатора из указанного буфера в случае удачи, иначе -1.

## GetData

Получает данные из буфера индикатора по начальному и конечному времени.

```
int GetData(
    const datetime start_time,      // начальное время
    const datetime stop_time,        // конечное время
    const int buffer_num,           // номер буфера
    double& buffer[]               // массив
) const
```

**Параметры***start\_time***[in]** Время начального элемента буфера индикатора.*stop\_time***[in]** Время конечного элемента буфера индикатора.*buffer\_num***[in]** Номер буфера индикатора.*buffer***[in]** Ссылка на массив для размещения данных.**Возвращаемое значение**

Количество полученных значений индикатора из указанного буфера в случае удачи, иначе -1.



## Refresh

Обновляет данные индикатора. Рекомендуется вызывать перед использованием [GetData\(\)](#).

```
virtual void Refresh(
    const int   flags=OBJ_ALL_PERIODS           // флаги
)
```

### Параметры

*flags*=OBJ\_ALL\_PERIODS

[in] Флаги обновления таймфреймов.

## Minimum

Получает индекс минимального элемента указанного буфера, в указанном интервале.

```
int Minimum(
    const int buffer_num,           // номер буфера
    const int start,                // стартовый индекс
    const int count                 // количество
) const
```

### Параметры

*buffer\_num*

[in] Номер буфера, для поиска значения.

*start*

[in] Стартовый индекс интервала поиска.

*count*

[in] Размер интервала поиска (количество элементов).

### Возвращаемое значение

Индекс минимального элемента указанного буфера, в указанном интервале.

## MinValue

Получает значение минимального элемента указанного буфера, в указанном интервале.

```
double MinValue(
    const int    buffer_num,      // номер буфера
    const int    start,          // стартовый индекс
    const int    count,          // количество
    int&        index           // ссылка
) const
```

### Параметры

*buffer\_num*

[in] Номер буфера, для поиска значения.

*start*

[in] Стартовый индекс интервала поиска.

*count*

[in] Размер интервала поиска (количество элементов).

*index*

[out] Ссылка на переменную, для размещения значения индекса найденного элемента.

### Возвращаемое значение

Значение минимального элемента указанного буфера, в указанном интервале.

### Примечание

Индекс найденного элемента помещается по ссылке *index*.

## Maximum

Получает индекс максимального элемента указанного буфера, в указанном интервале.

```
int Maximum(
    const int  buffer_num,      // номер буфера
    const int  start,          // стартовый индекс
    const int  count            // количество
) const
```

### Параметры

*buffer\_num*

[in] Номер буфера, для поиска значения.

*start*

[in] Стартовый индекс интервала поиска.

*count*

[in] Размер интервала поиска (количество элементов).

### Возвращаемое значение

Индекс максимального элемента указанного буфера, в указанном интервале.

## MaxValue

Получает значение максимального элемента указанного буфера, в указанном интервале.

```
double MaxValue(  
    const int    buffer_num,      // номер буфера  
    const int    start,          // стартовый индекс  
    const int    count,          // количество  
    int&        index           // ссылка  
) const
```

### Параметры

*buffer\_num*

[in] Номер буфера, для поиска значения.

*start*

[in] Стартовый индекс интервала поиска.

*count*

[in] Размер интервала поиска (количество элементов).

*index*

[out] Ссылка на переменную, для размещения значения индекса найденного элемента.

### Возвращаемое значение

Значение максимального элемента указанного буфера, в указанном интервале.

### Примечание

Индекс найденного элемента помещается по ссылке *index*.

## MethodDescription

Преобразует в строку значение перечисления [ENUM\\_MA\\_METHOD](#).

```
string MethodDescription(
    const int val        // значение
) const
```

### Параметры

*val*

[in] Значение для преобразования.

### Возвращаемое значение

Строка, соответствующая значению перечисления [ENUM\\_MA\\_METHOD](#).

## PriceDescription

Преобразует в строку значение перечисления [ENUM\\_APPLIED\\_PRICE](#).

```
string PriceDescription(
    const int val        // значение
) const
```

### Параметры

*val*

[in] Значение для преобразования.

### Возвращаемое значение

Строка, соответствующая значению перечисления [ENUM\\_APPLIED\\_PRICE](#).

## VolumeDescription

Преобразует в строку значение перечисления [ENUM\\_APPLIED\\_VOLUME](#).

```
string VolumeDescription(
    const int val        // значение
) const
```

### Параметры

*val*

[in] Значение для преобразования.

### Возвращаемое значение

Строка, соответствующая значению перечисления [ENUM\\_APPLIED\\_VOLUME](#).

## AddToChart

Добавляет индикатор на график.

```
bool AddToChart(
    const long chart,           // идентификатор графика
    const int subwin            // индекс подокна
)
```

### Параметры

*chart*

[in] Идентификатор графика.

*subwin*

[in] Индекс подокна графика.

### Возвращаемое значение

true - в случае удачи, false - если не удалось добавить индикатор на график.

## DeleteFromChart

Удаляет индикатор с графика.

```
bool DeleteFromChart(
    const long chart,           // идентификатор графика
    const int subwin            // индекс подокна
)
```

### Параметры

*chart*

[in] Идентификатор графика.

*subwin*

[in] Индекс подокна графика.

### Возвращаемое значение

true - в случае удачи, false - если не удалось удалить индикатор с графика.

## Класс CIndicators

Класс CIndicators является классом для коллекционирования экземпляров классов таймсерий и технических индикаторов.

### Описание

Класс CIndicators обеспечивает создание экземпляров классов технических индикаторов, их хранение и управление ими (синхронизацию данных, управление хэндлами и памятью).

### Декларация

```
class CIndicators: public CArrayObj
```

### Заголовок

```
#include <Indicators\Indicators.mqh>
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает индикатор
<a href="#">Обновление данных</a>	
<a href="#">Refresh</a>	Обновляет данные индикатора

## Create

Создает индикатор указанного типа с указанными параметрами.

```
CIndicator* Create(
    const string      symbol,      // имя символа
    const ENUM_TIMEFRAMES period,   // период
    const ENUM_INDICATOR type,     // тип
    const int         count,       // количество параметров
    const MqlParam&  params[]    // массив параметров
)
```

### Параметры

*symbol*

[in] Имя символа индикатора.

*period*

[in] Таймфрейм индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*type*

[in] Тип индикатора (значение перечисления [ENUM\\_INDICATOR](#)).

*count*

[in] Количество параметров для создания индикатора.

*params*

[in] Ссылка на массив параметров для создания индикатора.

### Возвращаемое значение

Указатель на созданный индикатор - в случае удачи, [NULL](#) - если не удалось создать индикатор.

## Refresh

Обновляет данные всех таймсерий и технических индикаторов коллекции.

```
int Refresh()
```

### Возвращаемое значение

Флаги обновленных таймфреймов (формируется как флаги видимости объектов.)

## Классы для работы таймсериями

Этот раздел содержит описание классов для работы таймсериями и описание соответствующих компонентов стандартной библиотеки MQL5.

Класс	Описание
<a href="#">CiSpread</a>	Класс доступа к истории спредов
<a href="#">CiTime</a>	Класс доступа к времени открытия баров
<a href="#">CiTickVolume</a>	Класс доступа к тиковым объемам баров
<a href="#">CiRealVolume</a>	Класс доступа к реальным объемам баров
<a href="#">CiOpen</a>	Класс доступа к ценам открытия баров
<a href="#">CiHigh</a>	Класс доступа к максимальным ценам баров
<a href="#">CiLow</a>	Класс доступа к минимальным ценам баров
<a href="#">CiClose</a>	Класс доступа к ценам закрытия баров

## Класс CiSpread

Класс CiSpread является классом для доступа к сериям истории спредов.

### Описание

Класс CiSpread обеспечивает доступ к сериям истории спредов.

### Декларация

```
class CiSpread: public CSeries
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CiSpread
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает серию
<a href="#">BufferResize</a>	Устанавливает размеры буфера серии
Доступ к данным	
<a href="#">GetData</a>	Получает данные серии
Обновление данных	
<a href="#">Refresh</a>	Обновляет данные серии

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

## Create

Создает таймсерию для доступа к истории спредов с указанными параметрами.

```
bool Create(
    string          symbol,      // имя символа
    ENUM_TIMEFRAMES period      // период
)
```

### Параметры

*symbol*

[in] Имя символа таймсерии.

*period*

[in] Таймфрейм таймсерии (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать таймсерию.

## BufferResize

Устанавливает размеры буфера серии.

```
virtual void BufferResize(  
    int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буфера.

## GetData

Получает указанный элемент буфера серии.

```
int GetData(
    int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера серии, либо 0.

## GetData

Получает данные из буфера таймсерии по стартовой позиции и количеству.

```
int GetData(
    int start_pos,      // позиция
    int count,          // количество
    int& buffer        // массив
) const
```

### Параметры

*start\_pos*

[in] Стартовая позиция буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

>=0 - в случае удачи, -1 - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному времени и количеству.

```
int GetData(
    datetime start_time,   // начальное время
    int count,             // количество
    int& buffer           // массив
) const
```

### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

#### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному и конечному времени.

```
int GetData(
    datetime start_time,      // начальное время
    datetime stop_time,       // конечное время
    int& buffer             // массив
) const
```

#### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*stop\_time*

[in] Время конечного элемента буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

#### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## Refresh

Обновляет данные серии.

```
virtual void Refresh(  
    int flags // флаги  
)
```

### Параметры

*flags*

[in] Флаги таймфреймов.

## Класс CiTime

Класс CiTime является классом для доступа к сериям времени открытия баров.

### Описание

Класс CiTime обеспечивает доступ к сериям времени открытия баров.

### Декларация

```
class CiTime: public CSeries
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
      CSeries
        CiTime
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает серию
<a href="#">BufferResize</a>	Устанавливает размеры буфера серии
Доступ к данным	
<a href="#">GetData</a>	Получает данные серии
Обновление данных	
<a href="#">Refresh</a>	Обновляет данные серии

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

## Create

Создает таймсерию для доступа к времени открытия баров с указанными параметрами.

```
bool Create(  
    string          symbol,      // имя символа  
    ENUM_TIMEFRAMES period      // период  
)
```

### Параметры

*symbol*

[in] Имя символа таймсерии.

*period*

[in] Таймфрейм таймсерии (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать таймсерию.

## BufferResize

Устанавливает размеры буфера серии.

```
virtual void BufferResize(  
    int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буфера.

## GetData

Получает указанный элемент буфера серии.

```
datetime GetData(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера серии, либо 0.

## GetData

Получает данные из буфера таймсерии по стартовой позиции и количеству.

```
int GetData(
    int start_pos,        // позиция
    int count,            // количество
    long& buffer          // массив
) const
```

### Параметры

*start\_pos*

[in] Стартовая позиция буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

>=0 - в случае удачи, -1 - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному времени и количеству.

```
int GetData(
    datetime start_time,      // начальное время
    int count,                // количество
    long& buffer              // массив
) const
```

### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

#### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному и конечному времени.

```
int GetData(
    datetime start_time,      // начальное время
    datetime stop_time,       // конечное время
    long& buffer            // массив
) const
```

#### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*stop\_time*

[in] Время конечного элемента буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

#### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## Refresh

Обновляет данные серии.

```
virtual void Refresh(  
    int flags // флаги  
)
```

### Параметры

*flags*

[in] Флаги таймфреймов.

## Класс CiTickVolume

Класс CiTickVolume является классом для доступа к сериям тиковых объемов баров.

### Описание

Класс CiTickVolume обеспечивает доступ к сериям тиковых объемов баров.

### Декларация

```
class CiTickVolume: public CSeries
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CiTickVolume
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает серию
<a href="#">BufferResize</a>	Устанавливает размеры буфера серии
Доступ к данным	
<a href="#">GetData</a>	Получает данные серии
Обновление данных	
<a href="#">Refresh</a>	Обновляет данные серии

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

## Create

Создает таймсерию для доступа к тиковым объемам баров с указанными параметрами.

```
bool Create(  
    string          symbol,      // имя символа  
    ENUM_TIMEFRAMES period      // период  
)
```

### Параметры

*symbol*

[in] Имя символа таймсерии.

*period*

[in] Таймфрейм таймсерии (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать таймсерию.

## BufferResize

Устанавливает размеры буфера серии.

```
virtual void BufferResize(  
    int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буфера.

## GetData

Получает указанный элемент буфера серии.

```
long GetData(
    int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера серии, либо 0.

## GetData

Получает данные из буфера таймсерии по стартовой позиции и количеству.

```
int GetData(
    int start_pos,      // позиция
    int count,          // количество
    long& buffer        // массив
) const
```

### Параметры

*start\_pos*

[in] Стартовая позиция буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

>=0 - в случае удачи, -1 - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному времени и количеству.

```
int GetData(
    datetime start_time,   // начальное время
    int count,             // количество
    long& buffer          // массив
) const
```

### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

#### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному и конечному времени.

```
int GetData(
    datetime start_time,      // начальное время
    datetime stop_time,       // конечное время
    long& buffer            // массив
) const
```

#### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*stop\_time*

[in] Время конечного элемента буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

#### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## Refresh

Обновляет данные серии.

```
virtual void Refresh(
    int   flags        // флаги
)
```

### Параметры

*flags*

[in] Флаги таймфреймов.

## Класс CiRealVolume

Класс CiRealVolume является классом для доступа к сериям реальных объёмов баров.

### Описание

Класс CiRealVolume обеспечивает доступ к сериям реальных объёмов баров.

### Декларация

```
class CiRealVolume: public CSeries
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CiRealVolume
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает серию
<a href="#">BufferResize</a>	Устанавливает размеры буфера серии
Доступ к данным	
<a href="#">GetData</a>	Получает данные серии
Обновление данных	
<a href="#">Refresh</a>	Обновляет данные серии

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

## Create

Создает таймсерию для доступа к реальным объемам баров с указанными параметрами.

```
bool Create(  
    string          symbol,      // имя символа  
    ENUM_TIMEFRAMES period      // период  
)
```

### Параметры

*symbol*

[in] Имя символа таймсерии.

*period*

[in] Таймфрейм таймсерии (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать таймсерию.

## BufferResize

Устанавливает размеры буфера серии.

```
virtual void BufferResize(  
    int size // размер  
)
```

### Параметры

*size*

[in] Новый размер буфера.

## GetData

Получает указанный элемент буфера серии.

```
long GetData(
    int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера серии, либо 0.

## GetData

Получает данные из буфера таймсерии по стартовой позиции и количеству.

```
int GetData(
    int start_pos,      // позиция
    int count,          // количество
    long& buffer        // массив
) const
```

### Параметры

*start\_pos*

[in] Стартовая позиция буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

>=0 - в случае удачи, -1 - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному времени и количеству.

```
int GetData(
    datetime start_time,   // начальное время
    int count,             // количество
    long& buffer          // массив
) const
```

### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

#### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному и конечному времени.

```
int GetData(
    datetime start_time,           // начальное время
    datetime stop_time,            // конечное время
    long& buffer                 // массив
) const
```

#### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*stop\_time*

[in] Время конечного элемента буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

#### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## Refresh

Обновляет данные серии.

```
virtual void Refresh(
    int   flags        // флаги
)
```

### Параметры

*flags*

[in] Флаги таймфреймов.

## Класс CiOpen

Класс CiOpen является классом для доступа к сериям цен открытия баров.

### Описание

Класс CiOpen обеспечивает доступ к сериям цен открытия баров.

### Декларация

```
class CiOpen: public CPriceSeries
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CPriceSeries
        CiOpen
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает серию
Доступ к данным	
<a href="#">GetData</a>	Получает данные серии

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Методы унаследованные от CPriceSeries

[BufferResize](#), [MinIndex](#), [MinValue](#), [MaxIndex](#), [MaxValue](#), [GetData](#), [Refresh](#)

## Create

Создает таймсерию для доступа к ценам открытия баров с указанными параметрами.

```
bool Create(  
    string          symbol,      // имя символа  
    ENUM_TIMEFRAMES period      // период  
)
```

### Параметры

*symbol*

[in] Имя символа таймсерии.

*period*

[in] Таймфрейм таймсерии (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать таймсерию.

## GetData

Получает данные из буфера таймсерии по стартовой позиции и количеству.

```
int GetData(
    int      start_pos,      // позиция
    int      count,          // количество
    double& buffer          // массив
) const
```

### Параметры

*start\_pos*

[in] Стартовая позиция буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

>=0 - в случае удачи, -1 - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному времени и количеству.

```
int GetData(
    datetime start_time,      // начальное время
    int      count,          // количество
    double& buffer          // массив
) const
```

### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

>=0 - в случае удачи, -1 - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному и конечному времени.

```
int GetData(
    datetime start_time,      // начальное время
    datetime stop_time,       // конечное время
    double& buffer           // массив
) const
```

### Параметры

*start\_time*  
[in] Время начального элемента буфера таймсерии.

*stop\_time*  
[in] Время конечного элемента буфера таймсерии.

*buffer*  
[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## Класс CiHigh

Класс CiHigh является классом для доступа к сериям максимальных цен баров.

### Описание

Класс CiHigh обеспечивает доступ к сериям максимальных цен баров.

### Декларация

```
class CiHigh: public CPriceSeries
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CPriceSeries
        CiHigh
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает серию
Доступ к данным	
<a href="#">GetData</a>	Получает данные серии

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Методы унаследованные от CPriceSeries

[BufferResize](#), [MinIndex](#), [MinValue](#), [MaxIndex](#), [MaxValue](#), [GetData](#), [Refresh](#)

## Create

Создает таймсерию для доступа к максимальным ценам баров, с указанными параметрами.

```
bool Create(  
    string          symbol,      // имя символа  
    ENUM_TIMEFRAMES period      // период  
)
```

### Параметры

*symbol*

[in] Имя символа таймсерии.

*period*

[in] Таймфрейм таймсерии (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать таймсерию.

## GetData

Получает данные из буфера таймсерии по стартовой позиции и количеству.

```
int GetData(
    int      start_pos,      // позиция
    int      count,          // количество
    double& buffer          // массив
) const
```

### Параметры

*start\_pos*

[in] Стартовая позиция буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

>=0 - в случае удачи, -1 - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному времени и количеству.

```
int GetData(
    datetime start_time,      // начальное время
    int      count,          // количество
    double& buffer          // массив
) const
```

### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

>=0 - в случае удачи, -1 - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному и конечному времени.

```
int GetData(
    datetime start_time,      // начальное время
    datetime stop_time,       // конечное время
    double& buffer           // массив
) const
```

### Параметры

*start\_time*  
[in] Время начального элемента буфера таймсерии.

*stop\_time*  
[in] Время конечного элемента буфера таймсерии.

*buffer*  
[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## Класс CiLow

Класс CiLow является классом для доступа к сериям минимальных цен баров.

### Описание

Класс CiLow обеспечивает доступ к сериям минимальных цен баров.

### Декларация

```
class CiLow: public CPriceSeries
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CPriceSeries
        CiLow
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает серию
Доступ к данным	
<a href="#">GetData</a>	Получает данные серии

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Методы унаследованные от CPriceSeries

[BufferResize](#), [MinIndex](#), [MinValue](#), [MaxIndex](#), [MaxValue](#), [GetData](#), [Refresh](#)

## Create

Создает таймсерию для доступа к минимальным ценам баров с указанными параметрами.

```
bool Create(
    string          symbol,      // имя символа
    ENUM_TIMEFRAMES period      // период
)
```

### Параметры

*symbol*

[in] Имя символа таймсерии.

*period*

[in] Таймфрейм таймсерии (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать таймсерию.

## GetData

Получает данные из буфера таймсерии по стартовой позиции и количеству.

```
int GetData(
    int      start_pos,      // позиция
    int      count,          // количество
    double& buffer          // массив
) const
```

### Параметры

*start\_pos*

[in] Стартовая позиция буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному времени и количеству.

```
int GetData(
    datetime start_time,      // начальное время
    int      count,          // количество
    double& buffer          // массив
) const
```

### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному и конечному времени.

```
int GetData(
    datetime start_time,      // начальное время
    datetime stop_time,       // конечное время
    double& buffer           // массив
) const
```

### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*stop\_time*

[in] Время конечного элемента буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

>=0 - в случае удачи, -1 - если не удалось получить данные.

## Класс CiClose

Класс CiClose является классом для доступа к сериям цен закрытия баров.

### Описание

Класс CiClose обеспечивает доступ к сериям цен закрытия баров.

### Декларация

```
class CiClose: public CPriceSeries
```

### Заголовок

```
#include <Indicators\TimeSeries.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CPriceSeries
        CiClose
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает серию
Доступ к данным	
<a href="#">GetData</a>	Получает данные серии

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Type](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

Методы унаследованные от CPriceSeries

[BufferResize](#), [MinIndex](#), [MinValue](#), [MaxIndex](#), [MaxValue](#), [GetData](#), [Refresh](#)

## Create

Создает таймсерию для доступа к ценам закрытия баров, с указанными параметрами.

```
bool Create(  
    string          symbol,      // имя символа  
    ENUM_TIMEFRAMES period      // период  
)
```

### Параметры

*symbol*

[in] Имя символа таймсерии.

*period*

[in] Таймфрейм таймсерии (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать таймсерию.

## GetData

Получает данные из буфера таймсерии по стартовой позиции и количеству.

```
int GetData(
    int      start_pos,      // позиция
    int      count,          // количество
    double& buffer          // массив
) const
```

### Параметры

*start\_pos*

[in] Стартовая позиция буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному времени и количеству.

```
int GetData(
    datetime start_time,      // начальное время
    int      count,          // количество
    double& buffer          // массив
) const
```

### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*count*

[in] Количество элементов буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

$\geq 0$  - в случае удачи,  $-1$  - если не удалось получить данные.

## GetData

Получает данные из буфера таймсерии по начальному и конечному времени.

```
int GetData(
    datetime start_time,      // начальное время
    datetime stop_time,       // конечное время
    double& buffer           // массив
) const
```

### Параметры

*start\_time*

[in] Время начального элемента буфера таймсерии.

*stop\_time*

[in] Время конечного элемента буфера таймсерии.

*buffer*

[in] Ссылка на массив для размещения данных.

### Возвращаемое значение

>=0 - в случае удачи, -1 - если не удалось получить данные.

## Группа базовых и вспомогательных классов технических индикаторов и таймсерий

Этот раздел содержит детали работы с группой базовых и вспомогательных классов технических индикаторов и таймсерий и описание соответствующих компонентов стандартной библиотеки MQL5.

Класс/группа	Описание
<a href="#">CiADX</a>	Индикатор "Average Directional Index"
<a href="#">CiADXWilder</a>	Индикатор "Average Directional Index by Welles Wilder"
<a href="#">CiBands</a>	Индикатор "Bollinger Bands®"
<a href="#">CiEnvelopes</a>	Индикатор "Envelopes"
<a href="#">CiIchimoku</a>	Индикатор "Ichimoku Kinko Hyo"
<a href="#">CiMA</a>	Индикатор "Moving Average"
<a href="#">CiSAR</a>	Индикатор "Parabolic Stop And Reverse System"
<a href="#">CiStdDev</a>	Индикатор "Standard Deviation"
<a href="#">CiDEMA</a>	Индикатор "Double Exponential Moving Average"
<a href="#">CiTEMA</a>	Индикатор "Triple Exponential Moving Average"
<a href="#">CiFrAMA</a>	Индикатор "Fractal Adaptive Moving Average"
<a href="#">CiAMA</a>	Индикатор "Adaptive Moving Average"
<a href="#">CiVIDyA</a>	Индикатор "Variable Index Dynamic Average"

## Класс CiADX

Класс CiADX является классом для работы с техническим индикатором "Average Directional Index".

### Описание

Класс CiADX обеспечивает создание, настройку и доступ к данным индикатора "Average Directional Index".

### Декларация

```
class CiADX: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiADX
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера основной линии
<a href="#">Plus</a>	Получает данные буфера линии +DI
<a href="#">Minus</a>	Получает данные буфера линии -DI
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period        // период усреднения
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера основной линии по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера основной линии.

### Возвращаемое значение

Элемент буфера основной линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Plus

Получает элемент буфера линии +DI по указанному индексу.

```
double Plus(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии +DI.

### Возвращаемое значение

Элемент буфера линии +DI по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Minus

Получает элемент буфера линии -DI по указанному индексу.

```
double Minus(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии -DI.

### Возвращаемое значение

Элемент буфера линии -DI по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiADX - [IND\\_ADX](#)).

## Класс CiADXWilder

Класс CiADXWilder является классом для работы с техническим индикатором "Average Directional Index by Welles Wilder".

### Описание

Класс CiADXWilder обеспечивает создание, настройку и доступ к данным индикатора "Average Directional Index by Welles Wilder".

### Декларация

```
class CiADXWilder: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiADXWilder
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера основной линии
<a href="#">Plus</a>	Получает данные буфера линии +DI
<a href="#">Minus</a>	Получает данные буфера линии -DI
<b>Ввод/вывод</b>	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period        // период усреднения
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера основной линии по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера основной линии.

### Возвращаемое значение

Элемент буфера основной линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Plus

Получает элемент буфера линии +DI по указанному индексу.

```
double Plus(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии +DI.

### Возвращаемое значение

Элемент буфера линии +DI по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Minus

Получает элемент буфера линии -DI по указанному индексу.

```
double Minus(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии -DI.

### Возвращаемое значение

Элемент буфера линии -DI по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiADXWilder - [IND\\_ADXW](#)).

## Класс CiBands

Класс CiBands является классом для работы с техническим индикатором "Bollinger Bands®".

### Описание

Класс CiBands обеспечивает создание, настройку и доступ к данным индикатора "Bollinger Bands".

### Декларация

```
class CiBands: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiBands
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">MaShift</a>	Получает смещение по оси цен
<a href="#">Deviation</a>	Получает значение девиации
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Base</a>	Получает данные буфера основной линии
<a href="#">Upper</a>	Получает данные буфера верхней линии
<a href="#">Lower</a>	Получает данные буфера нижней линии
<b>Ввод/вывод</b>	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## MaShift

Получает смещение по оси цен.

```
int MaShift() const
```

### Возвращаемое значение

Смещение по оси цен, назначенное при создании индикатора.

## Deviation

Получает значение девиации.

```
double Deviation() const
```

### Возвращаемое значение

Значение девиации, назначенное при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ma_shift,         // смещение
    double          deviation,        // девиация
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*ma\_shift*

[in] Смещение индикатора по оси цен.

*deviation*

[in] Девиация индикатора.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Base

Получает элемент буфера основной линии по указанному индексу.

```
double Base(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера основной линии.

### Возвращаемое значение

Элемент буфера основной линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Upper

Получает элемент буфера верхней линии по указанному индексу.

```
double Upper(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера верхней линии.

### Возвращаемое значение

Элемент буфера верхней линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Lower

Получает элемент буфера нижней линии по указанному индексу.

```
double Lower(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера нижней линии.

### Возвращаемое значение

Элемент буфера нижней линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiBands - [IND\\_BANDS](#)).

## Класс CiEnvelopes

Класс CiEnvelopes является классом для работы с техническим индикатором "Envelopes".

### Описание

Класс CiEnvelopes обеспечивает создание, настройку и доступ к данным индикатора "Envelopes".

### Декларация

```
class CiEnvelopes: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
      CiEnvelopes
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">MaShift</a>	Получает смещение по оси цен
<a href="#">MaMethod</a>	Получает метод усреднения
<a href="#">Deviation</a>	Получает значение девиации
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Upper</a>	Получает данные буфера верхней линии
<a href="#">Lower</a>	Получает данные буфера нижней линии
Ввод/вывод	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## MaShift

Получает смещение по оси цен.

```
int MaShift() const
```

### Возвращаемое значение

Смещение по оси цен, назначенное при создании индикатора.

## MaMethod

Получает метод усреднения.

```
ENUM_MA_METHOD MaMethod() const
```

### Возвращаемое значение

Метод усреднения, назначенный при создании индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

## Deviation

Получает значение девиации.

```
double Deviation() const
```

### Возвращаемое значение

Значение девиации, назначенное при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ma_shift,         // смещение
    ENUM_MA_METHOD ma_method,        // метод усреднения
    int             applied,          // тип цены, хэндл
    double          deviation        // девиация
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*ma\_shift*

[in] Смещение индикатора по оси цен.

*ma\_method*

[in] Метод усреднения индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

*deviation*

[in] Девиация индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Upper

Получает элемент буфера верхней линии по указанному индексу.

```
double Upper(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера верхней линии.

### Возвращаемое значение

Элемент буфера верхней линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Lower

Получает элемент буфера нижней линии по указанному индексу.

```
double Lower(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера нижней линии.

### Возвращаемое значение

Элемент буфера нижней линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiEnvelopes - [IND\\_ENVELOPES](#)).

## Класс Cilchimoku

Класс Cilchimoku является классом для работы с техническим индикатором "Ichimoku Kinko Hyo".

### Описание

Класс Cilchimoku обеспечивает создание, настройку и доступ к данным индикатора "Ichimoku Kinko Hyo".

### Декларация

```
class CiIchimoku: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    Cilchimoku
```

### Методы класса по группам

Атрибуты	
<a href="#">TenkanSenPeriod</a>	Получает период TenkanSen
<a href="#">KijunSenPeriod</a>	Получает период KijunSen
<a href="#">SenkouSpanBPeriod</a>	Получает период SenkouSpanB
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">TenkanSen</a>	Получает данные буфера линии TenkanSen
<a href="#">KijunSen</a>	Получает данные буфера линии KijunSen
<a href="#">SenkouSpanA</a>	Получает данные буфера линии SenkouSpanA
<a href="#">SenkouSpanB</a>	Получает данные буфера линии SenkouSpanB
<a href="#">ChinkouSpan</a>	Получает данные буфера линии ChinkouSpan
<b>Ввод/вывод</b>	
<b>virtual <a href="#">Type</a></b>	Виртуальный метод идентификации

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)**Методы унаследованные от CArrayObj**[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreaterOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)**Методы унаследованные от CSeries**[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)**Методы унаследованные от CIndicator**[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## TenkanSenPeriod

Получает период TenkanSen.

```
int TenkanSenPeriod() const
```

### Возвращаемое значение

Период TenkanSen, назначенный при создании индикатора.

## KijunSenPeriod

Получает период KijunSen.

```
int KijunSenPeriod() const
```

### Возвращаемое значение

Период KijunSen, назначенный при создании индикатора.

## SenkouSpanBPeriod

Получает период SenkouSpanB.

```
int SenkouSpanBPeriod() const
```

### Возвращаемое значение

Период SenkouSpanB, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             tenkan_sen,        // период TenkanSen
    int             kijun_sen,         // период KijunSen
    int             senkou_span_b     // период SenkouSpanB
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*tenkan\_sen*

[in] Период TenkanSen индикатора.

*kijun\_sen*

[in] Период KijunSen индикатора.

*senkou\_span\_b*

[in] Период SenkouSpanB индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## TenkanSen

Получает элемент буфера линии TenkanSen по указанному индексу.

```
double TenkanSen(
    int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии TenkanSen.

### Возвращаемое значение

Элемент буфера линии TenkanSen по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## KijunSen

Получает элемент буфера линии KijunSen по указанному индексу.

```
double KijunSen(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии KijunSen.

### Возвращаемое значение

Элемент буфера линии KijunSen по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## SenkouSpanA

Получает элемент буфера линии SenkouSpanA по указанному индексу.

```
double SenkouSpanA(
    int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии SenkouSpanA.

### Возвращаемое значение

Элемент буфера линии SenkouSpanA по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## SenkouSpanB

Получает элемент буфера линии SenkouSpanB по указанному индексу.

```
double SenkouSpanB(
    int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии SenkouSpanB.

### Возвращаемое значение

Элемент буфера линии SenkouSpanB по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## ChinkouSpan

Получает элемент буфера линии ChinkouSpan по указанному индексу.

```
double ChinkouSpan(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии ChinkouSpan.

### Возвращаемое значение

Элемент буфера линии ChinkouSpan по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CIchimoku - [IND\\_ICHIMOKU](#)).

## Класс CiMA

Класс CiMA является классом для работы с техническим индикатором "Moving Average".

### Описание

Класс CiMA обеспечивает создание, настройку и доступ к данным индикатора "Moving Average".

### Декларация

```
class CiMA: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiMA
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">MaShift</a>	Получает смещение по оси цен
<a href="#">MaMethod</a>	Получает метод усреднения
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Main</a>	Получает данные буфера
Ввод/вывод	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## MaShift

Получает смещение по оси цен.

```
int MaShift() const
```

### Возвращаемое значение

Смещение по оси цен, назначенное при создании индикатора.

## MaMethod

Получает метод усреднения.

```
ENUM_MA_METHOD MaMethod() const
```

### Возвращаемое значение

Метод усреднения, назначенный при создании индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string      string,           // символ
    ENUM_TIMEFRAMES period,       // период
    int         ma_period,        // период усреднения
    int         ma_shift,          // смещение
    ENUM_MA_METHOD ma_method,     // метод усреднения
    int         applied           // тип цены, хэндл
)
```

### Параметры

*string*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*ma\_shift*

[in] Смещение индикатора по оси цен.

*ma\_method*

[in] Метод усреднения индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiMA - [IND\\_MA](#)).

## Класс CiSAR

Класс CiSAR является классом для работы с техническим индикатором "Parabolic Stop And Reverse System".

### Описание

Класс CiSAR обеспечивает создание, настройку и доступ к данным индикатора "Parabolic Stop And Reverse System".

### Декларация

```
class CiSAR: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiSAR
```

### Методы класса по группам

Атрибуты	
<a href="#">SarStep</a>	Получает шаг изменения цены
<a href="#">Maximum</a>	Получает максимальное значение шага
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## SarStep

Получает шаг изменения цены.

```
double SarStep() const
```

### Возвращаемое значение

Шаг изменения цены, указанный при создании индикатора.

## Maximum

Получает максимальное значение шага.

```
double Maximum() const
```

### Возвращаемое значение

Максимальное значение шага, указанное при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,      // символ
    ENUM_TIMEFRAMES period,     // период
    double          step,        // шаг
    double          maximum      // коэффициент
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*step*

[in] Шаг увеличения скорости.

*maximum*

[in] Коэффициент следования за ценой.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiSAR - [IND\\_SAR](#)).

## Класс CiStdDev

Класс CiStdDev является классом для работы с техническим индикатором "Standard Deviation".

### Описание

Класс CiStdDev обеспечивает создание, настройку и доступ к данным индикатора "Standard Deviation".

### Декларация

```
class CiStdDev: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiStdDev
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">MaShift</a>	Получает смещение по оси цен
<a href="#">MaMethod</a>	Получает метод усреднения
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Main</a>	Получает данные буфера
Ввод/вывод	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

**Методы унаследованные от CArray**

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## MaShift

Получает смещение по оси цен.

```
int MaShift() const
```

### Возвращаемое значение

Смещение по оси цен, назначенное при создании индикатора.

## MaMethod

Получает метод усреднения.

```
ENUM_MA_METHOD MaMethod() const
```

### Возвращаемое значение

Метод усреднения, назначенный при создании индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // цвет // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ma_shift,         // смещение
    ENUM_MA_METHOD ma_method,        // метод усреднения
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*ma\_shift*

[in] Смещение индикатора по оси цен.

*ma\_method*

[in] Метод усреднения индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiStdDev - [IND\\_STDDEV](#)).

## Класс CiDEMA

Класс CiDEMA является классом для работы с техническим индикатором "Double Exponential Moving Average".

### Описание

Класс CiDEMA обеспечивает создание, настройку и доступ к данным индикатора "Double Exponential Moving Average".

### Декларация

```
class CiDEMA: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiDEMA
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">IndShift</a>	Получает смещение по оси цен
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

**Методы унаследованные от CArray**

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## IndShift

Получает смещение по оси цен.

```
int IndShift() const
```

### Возвращаемое значение

Смещение по оси цен, назначенное при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          string,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ind_shift,        // смещение
    int             applied           // тип цены, хэндл
)
```

### Параметры

*string*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*ind\_shift*

[in] Смещение индикатора по оси цен.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiDEMA - [IND\\_DEMA](#)).

## Класс CiTEMA

Класс CiTEMA является классом для работы с техническим индикатором "Triple Exponential Moving Average".

### Описание

Класс CiTEMA обеспечивает создание, настройку и доступ к данным индикатора "Triple Exponential Moving Average".

### Декларация

```
class CiTEMA: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiTEMA
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">IndShift</a>	Получает смещение по оси цен
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

Prev, Prev, Next, Next, [Compare](#)

**Методы унаследованные от CArray**

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## IndShift

Получает смещение по оси цен.

```
int IndShift() const
```

### Возвращаемое значение

Смещение по оси цен, назначенное при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ma_shift,         // смещение
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*ma\_shift*

[in] Смещение индикатора по оси цен.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiTEMA - [IND\\_TEMA](#)).

## Класс CiFrAMA

Класс CiFrAMA является классом для работы с техническим индикатором " Fractal Adaptive Moving Average".

### Описание

Класс CiFrAMA обеспечивает создание, настройку и доступ к данным индикатора " Fractal Adaptive Moving Average".

### Декларация

```
class CiFrAMA: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiFrAMA
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">IndShift</a>	Получает смещение по оси цен
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

Prev, Prev, Next, Next, [Compare](#)

**Методы унаследованные от CArray**

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## IndShift

Получает смещение по оси цен.

```
int IndShift() const
```

### Возвращаемое значение

Смещение по оси цен, назначенное при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             ma_shift,         // смещение
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*ma\_shift*

[in] Смещение индикатора по оси цен.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiFrAMA- [IND\\_FRAMA](#)).

## Класс CiAMA

Класс CiAMA является классом для работы с техническим индикатором "Adaptive Moving Average".

### Описание

Класс CiAMA обеспечивает создание, настройку и доступ к данным индикатора "Adaptive Moving Average".

### Декларация

```
class CiAMA: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiAMA
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">FastEmaPeriod</a>	Получает период усреднения быстрой ЕМА
<a href="#">SlowEmaPeriod</a>	Получает период усреднения медленной ЕМА
<a href="#">IndShift</a>	Получает смещение по оси цен
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## FastEmaPeriod

Получает период усреднения быстрой ЕМА.

```
int FastEmaPeriod() const
```

### Возвращаемое значение

Период усреднения быстрой ЕМА, назначенный при создании индикатора.

## SlowEmaPeriod

Получает период усреднения медленной ЕМА.

```
int SlowEmaPeriod() const
```

### Возвращаемое значение

Период усреднения медленной ЕМА, назначенный при создании индикатора.

## IndShift

Получает смещение по оси цен.

```
int IndShift() const
```

### Возвращаемое значение

Смещение по оси цен, назначенное при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          string,           // символ
    ENUM_TIMEFRAMES period,         // период
    int             ma_period,       // период усреднения
    int             fast_ema_period, // период быстрой ЕМА
    int             slow_ema_period, // период медленной ЕМА
    int             ind_shift,        // смещение
    int             applied          // тип цены, хэндл
)
```

### Параметры

*string*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*fast\_ema\_period*

[in] Период усреднения быстрой ЕМА индикатора.

*slow\_ema\_period*

[in] Период усреднения медленной ЕМА индикатора.

*ind\_shift*

[in] Смещение индикатора по оси цен.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiAMA - [IND\\_AMA](#)).

## Класс CiVIDyA

Класс CiVIDyA является классом для работы с техническим индикатором "Variable Index Dynamic Average".

### Описание

Класс CiVIDyA обеспечивает создание, настройку и доступ к данным индикатора "Variable Index Dynamic Average".

### Декларация

```
class CiVIDyA: public CIndicator
```

### Заголовок

```
#include <Indicators\Trend.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiVIDyA
```

### Методы класса по группам

Атрибуты	
<a href="#">CmoPeriod</a>	Получает период Momentum
<a href="#">EmaPeriod</a>	Получает период сглаживания
<a href="#">IndShift</a>	Получает смещение по оси цен
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Main</a>	Получает данные буфера
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

**Методы унаследованные от CArray**

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## CmoPeriod

Закрывает график привязанный к экземпляру класса.

```
int CmoPeriod() const
```

## EmaPeriod

Получает имя символа графика.

```
int EmaPeriod() const
```

### Возвращаемое значение

Имя символа графика, привязанного к экземпляру класса. Если нет привязанного графика, возвращается "".

## IndShift

Получает смещение по оси цен.

```
int IndShift() const
```

### Возвращаемое значение

Смещение по оси цен, назначенное при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             cmo_period,       // период Momentum
    int             ema_period,        // период сглаживания
    int             ind_shift,         // смещение
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*cmo\_period*

[in] Период Momentum индикатора.

*ema\_period*

[in] Период сглаживания индикатора.

*ind\_shift*

[in] Смещение индикатора по оси цен.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiVIDyA - [IND\\_VIDYA](#)).

## Группа классов "Осциляторы"

Этот раздел содержит детали работы с группой классов "Осциляторы" и описание соответствующих компонентов стандартной библиотеки MQL5.

Класс/группа	Описание
<a href="#">CiATR</a>	Осциллятор "Average True Range"
<a href="#">CiBearsPower</a>	Осциллятор "Bears Power"
<a href="#">CiBullsPower</a>	Осциллятор "Bulls Power"
<a href="#">CiCCI</a>	Осциллятор "Commodity Channel Index"
<a href="#">CiChaikin</a>	Осциллятор "Chaikin Oscillator"
<a href="#">CiDeMarker</a>	Осциллятор "DeMarker"
<a href="#">CiForce</a>	Осциллятор "Force Index"
<a href="#">CiMACD</a>	Осциллятор "Moving Averages Convergence-Divergence"
<a href="#">CiMomentum</a>	Осциллятор "Momentum"
<a href="#">CiOsMA</a>	Осциллятор "Moving Average of Oscillator (MACD histogram)"
<a href="#">CiRSI</a>	Осциллятор "Relative Strength Index"
<a href="#">CiRVI</a>	Осциллятор "Relative Vigor Index"
<a href="#">CiStochastic</a>	Осциллятор "Stochastic Oscillator"
<a href="#">CiWPR</a>	Осциллятор "Williams' Percent Range"
<a href="#">CiTriX</a>	Осциллятор "Triple Exponential Moving Averages Oscillator"

## Класс CiATR

Класс CiATR является классом для работы с техническим индикатором "Average True Range".

### Описание

Класс CiATR обеспечивает создание, настройку и доступ к данным индикатора "Average True Range".

### Декларация

```
class CiATR: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscillators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
      CiATR
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period        // период усреднения
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiATR - [IND\\_ATR](#)).

## Класс CiBearsPower

Класс CiBearsPower является классом для работы с техническим индикатором "Bears Power".

### Описание

Класс CiBearsPower обеспечивает создание, настройку и доступ к данным индикатора "Bears Power".

### Декларация

```
class CiBearsPower: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscillators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiBearsPower
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period        // период усреднения
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiBearsPower - [IND\\_BEARS](#)).

## Класс CiBullsPower

Класс CiBullsPower является классом для работы с техническим индикатором "Bulls Power".

### Описание

Класс CiBullsPower обеспечивает создание, настройку и доступ к данным индикатора "Bulls Power".

### Декларация

```
class CiBullsPower: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscilators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
      CiBullsPower
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),  
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),  
[GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),  
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period        // период усреднения
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiBullsPower - [IND\\_BULLS](#)).

## Класс CiCCI

Класс CiCCI является классом для работы с техническим индикатором "Commodity Channel Index".

### Описание

Класс CiCCI обеспечивает создание, настройку и доступ к данным индикатора "Commodity Channel Index".

### Декларация

```
class CiCCI: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscillators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiCCI
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiCCI - [IND\\_CCI](#)).

## Класс CiChaikin

Класс CiChaikin является классом для работы с техническим индикатором "Chaikin Oscillator".

### Описание

Класс CiChaikin обеспечивает создание, настройку и доступ к данным индикатора "Chaikin Oscillator".

### Декларация

```
class CiChaikin: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscillators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiChaikin
```

### Методы класса по группам

Атрибуты	
<a href="#">FastMaPeriod</a>	Получает период усреднения быстрой МА
<a href="#">SlowMaPeriod</a>	Получает период усреднения медленной МА
<a href="#">MaMethod</a>	Получает период усреднения
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Main</a>	Получает данные буфера
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

**Методы унаследованные от CArray**

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## FastMaPeriod

Получает период усреднения быстрой ЕМА.

```
int FastMaPeriod() const
```

### Возвращаемое значение

Период усреднения быстрой ЕМА, назначенный при создании индикатора.

## SlowMaPeriod

Получает период усреднения медленной ЕМА.

```
int SlowMaPeriod() const
```

### Возвращаемое значение

Период усреднения медленной ЕМА, назначенный при создании индикатора.

## MaMethod

Получает метод усреднения.

```
ENUM_MA_METHOD MaMethod() const
```

### Возвращаемое значение

Метод усреднения, назначенный при создании индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

## Applied

Получает объект (тип объемов) применения.

```
ENUM_APPLIED_VOLUME Applied() const
```

### Возвращаемое значение

Объект (тип объемов) применения, назначенный при создании индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             fast_ma_period,   // период быстрой ЕМА
    int             slow_ma_period,   // период медленной ЕМА
    ENUM_MA_METHOD ma_method,        // метод усреднения
    ENUM_APPLIED_VOLUME applied       // тип объемов
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*fast\_ma\_period*

[in] Период усреднения быстрой ЕМА индикатора.

*slow\_ma\_period*

[in] Период усреднения медленной ЕМА индикатора.

*ma\_method*

[in] Метод усреднения индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

*applied*

[in] Объект (тип объемов) применения индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiChaikin - [IND\\_CHAIKIN](#)).

## Класс CiDeMarker

Класс CiDeMarker является классом для работы с техническим индикатором "DeMarker".

### Описание

Класс CiDeMarker обеспечивает создание, настройку и доступ к данным индикатора "DeMarker".

### Декларация

```
class CiDeMarker: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscilators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiDeMarker
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Main</a>	Получает данные буфера
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),  
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),  
[GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),  
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period        // период усреднения
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiDeMarker - [IND\\_DEMARKER](#)).

## Класс CiForce

Класс CiForce является классом для работы с техническим индикатором "Force Index".

### Описание

Класс CiForce обеспечивает создание, настройку и доступ к данным индикатора "Force Index".

### Декларация

```
class CiForce: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscillators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
      CiForce
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">MaMethod</a>	Получает метод усреднения
<a href="#">Applied</a>	Получает объект (тип объемов) применения
Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Main</a>	Получает данные буфера
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## MaMethod

Получает метод усреднения.

```
ENUM_MA_METHOD MaMethod() const
```

### Возвращаемое значение

Метод усреднения, назначенный при создании индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

## Applied

Получает объект (тип объемов) применения.

```
ENUM_APPLIED_VOLUME Applied() const
```

### Возвращаемое значение

Объект (тип объемов) применения, назначенный при создании индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    ENUM_MA_METHOD ma_method,        // метод усреднения
    ENUM_APPLIED_VOLUME applied       // тип объемов
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*ma\_method*

[in] Метод усреднения индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

*applied*

[in] Объект (тип объемов) применения индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiForce - [IND\\_FORCE](#)).

## Класс CiMACD

Класс CiMACD является классом для работы с техническим индикатором "Moving Averages Convergence-Divergence".

### Описание

Класс CiMACD обеспечивает создание, настройку и доступ к данным индикатора "Moving Averages Convergence-Divergence".

### Декларация

```
class CiMACD: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscilators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiMACD
```

### Методы класса по группам

Атрибуты	
<a href="#">FastEmaPeriod</a>	Получает период усреднения быстрой ЕМА
<a href="#">SlowEmaPeriod</a>	Получает период усреднения медленной ЕМА
<a href="#">SignalPeriod</a>	Получает период усреднения сигнальной линии
<a href="#">Applied</a>	Получает объект (тип объемов) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера основной линии
<a href="#">Signal</a>	Получает данные буфера сигнальной линии
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)**Методы унаследованные от CArrayObj**[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreaterOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)**Методы унаследованные от CSeries**[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)**Методы унаследованные от CIndicator**[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## FastEmaPeriod

Получает период усреднения быстрой ЕМА.

```
int FastEmaPeriod() const
```

### Возвращаемое значение

Период усреднения быстрой ЕМА, назначенный при создании индикатора.

## SlowEmaPeriod

Получает период усреднения медленной ЕМА.

```
int SlowEmaPeriod() const
```

### Возвращаемое значение

Период усреднения медленной ЕМА, назначенный при создании индикатора.

## SignalPeriod

Получает период усреднения сигнальной линии.

```
int SignalPeriod() const
```

### Возвращаемое значение

Период усреднения сигнальной линии, назначенный при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             fast_ema_period,   // период быстрой ЕМА
    int             slow_ema_period,   // период медленной ЕМА
    int             signal_period,    // период сигнала
    int             applied          // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*fast\_ema\_period*

[in] Период усреднения быстрой ЕМА индикатора.

*slow\_ema\_period*

[in] Период усреднения медленной ЕМА индикатора.

*signal\_period*

[in] Период усреднения сигнальной линии индикатора.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера основной линии по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера основной линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Signal

Получает элемент буфера сигнальной линии по указанному индексу.

```
double Signal(
    int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера сигнальной линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiMACD - [IND\\_MACD](#)).

## Класс CiMomentum

Класс CiMomentum является классом для работы с техническим индикатором "Momentum".

### Описание

Класс CiMomentum обеспечивает создание, настройку и доступ к данным индикатора "Momentum".

### Декларация

```
class CiMomentum: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscilators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiMomentum
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">Applied</a>	Получает объект (тип объемов) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiMomentum - [IND\\_MOMENTUM](#)).

## Класс CiOsMA

Класс CiOsMA является классом для работы с техническим индикатором "Moving Average of Oscillator (MACD histogram)".

### Описание

Класс CiOsMA обеспечивает создание, настройку и доступ к данным индикатора "Moving Average of Oscillator (MACD histogram)".

### Декларация

```
class CiOsMA: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscilators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiOsMA
```

### Методы класса по группам

Атрибуты	
<a href="#">FastEmaPeriod</a>	Получает период усреднения быстрой ЕМА
<a href="#">SlowEmaPeriod</a>	Получает период усреднения медленной ЕМА
<a href="#">SignalPeriod</a>	Получает период усреднения сигнальной линии
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
<a href="#">Создание</a>	
<a href="#">Create</a>	Создает индикатор
<a href="#">Доступ к данным</a>	
<a href="#">Main</a>	Получает данные буфера
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## FastEmaPeriod

Получает период усреднения быстрой ЕМА.

```
int FastEmaPeriod() const
```

### Возвращаемое значение

Период усреднения быстрой ЕМА, назначенный при создании индикатора.

## SlowEmaPeriod

Получает период усреднения медленной ЕМА.

```
int SlowEmaPeriod() const
```

### Возвращаемое значение

Период усреднения медленной ЕМА, назначенный при создании индикатора.

## SignalPeriod

Получает период усреднения сигнальной линии.

```
int SignalPeriod() const
```

### Возвращаемое значение

Период усреднения сигнальной линии, назначенный при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             fast_ema_period,   // период быстрой ЕМА
    int             slow_ema_period,   // период медленной ЕМА
    int             signal_period,    // период сигнала
    int             applied          // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*fast\_ema\_period*

[in] Период усреднения быстрой ЕМА индикатора.

*slow\_ema\_period*

[in] Период усреднения медленной ЕМА индикатора.

*signal\_period*

[in] Период усреднения сигнальной линии индикатора.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiOsMA - [IND\\_OSMA](#)).

## Класс CiRSI

Класс CiRSI является классом для работы с техническим индикатором "Relative Strength Index".

### Описание

Класс CiRSI обеспечивает создание, настройку и доступ к данным индикатора "Relative Strength Index".

### Декларация

```
class CiRSI: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscillators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
      CiRSI
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiRSI - [IND\\_RSI](#)).

## Класс CiRVI

Класс CiRVI является классом для работы с техническим индикатором "Relative Vigor Index".

### Описание

Класс CiRVI обеспечивает создание, настройку и доступ к данным индикатора "Relative Vigor Index".

### Декларация

```
class CiRVI: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscillators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiRVI
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера основной линии
<a href="#">Signal</a>	Получает данные буфера сигнальной линии
<b>Ввод/вывод</b>	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period        // период усреднения
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера основной линии по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера основной линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Signal

Получает элемент буфера сигнальной линии по указанному индексу.

```
double Signal(
    int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера сигнальной линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiRVI - [IND\\_RVI](#)).

## Класс CiStochastic

Класс CiStochastic является классом для работы с техническим индикатором "Stochastic Oscillator".

### Описание

Класс CiStochastic обеспечивает создание, настройку и доступ к данным индикатора "Stochastic Oscillator".

### Декларация

```
class CiStochastic: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscillators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiStochastic
```

### Методы класса по группам

Атрибуты	
<a href="#">Kperiod</a>	Получает период усреднения %K
<a href="#">Dperiod</a>	Получает период усреднения %D
<a href="#">Slowing</a>	Получает период замедления
<a href="#">MaMethod</a>	Получает метод усреднения
<a href="#">PriceField</a>	Получает объект (Low/High или Close/Close) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера основной линии
<a href="#">Signal</a>	Получает данные буфера сигнальной линии
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)**Методы унаследованные от CArrayObj**[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)**Методы унаследованные от CSeries**[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)**Методы унаследованные от CIndicator**[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## Kperiod

Получает период усреднения %K.

```
int Kperiod() const
```

### Возвращаемое значение

Период усреднения %K, назначенный при создании индикатора.

## Dperiod

Получает период усреднения %D.

```
int Dperiod() const
```

### Возвращаемое значение

Период усреднения %D, назначенный при создании индикатора.

## Slowing

Получает период замедления.

```
int Slowing() const
```

### Возвращаемое значение

Период замедления, назначенный при создании индикатора.

## MaMethod

Получает метод усреднения.

```
ENUM_MA_METHOD MaMethod() const
```

### Возвращаемое значение

Метод усреднения, назначенный при создании индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

## PriceField

Получает объект (Low/High или Close/Close) применения.

```
ENUM_STO_PRICE PriceField() const
```

### Возвращаемое значение

Объект (Low/High или Close/Close) применения, назначенный при создании индикатора (значение перечисления [ENUM\\_STO\\_PRICE](#)).

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             Kperiod,          // период %K
    int             Dperiod,          // период %D
    int             slowing,           // период замедления
    ENUM_MA_METHOD ma_method,        // метод усреднения
    ENUM_STO_PRICE  price_field     // применение
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*Kperiod*

[in] Период усреднения %K индикатора.

*Dperiod*

[in] Период усреднения %D индикатора.

*slowing*

[in] Период замедления индикатора.

*ma\_method*

[in] Метод усреднения индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

*price\_field*

[in] Объект (Low/High или Close/Close) применения индикатора (значение перечисления [ENUM\\_STO\\_PRICE](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера основной линии по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера основной линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Signal

Получает элемент буфера сигнальной линии по указанному индексу.

```
double Signal(
    int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера сигнальной линии по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiStochastic - [IND\\_STOCHASTIC](#)).

## Класс CiTriX

Класс CiTriX является классом для работы с техническим индикатором "Triple Exponential Moving Averages Oscillator".

### Описание

Класс CiTriX обеспечивает создание, настройку и доступ к данным индикатора "Triple Exponential Moving Averages Oscillator".

### Декларация

```
class CiTriX: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscilators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiTriX
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiTriX - [IND\\_TRIX](#)).

## Класс CiWPR

Класс CiWPR является классом для работы с техническим индикатором "Williams' Percent Range".

### Описание

Класс CiWPR обеспечивает создание, настройку и доступ к данным индикатора "Williams' Percent Range".

### Декларация

```
class CiWPR: public CIndicator
```

### Заголовок

```
#include <Indicators\Oscillators.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiWPR
```

### Методы класса по группам

Атрибуты	
<a href="#">CalcPeriod</a>	Получает период расчета
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## CalcPeriod

Получает период расчета.

```
int CalcPeriod() const
```

### Возвращаемое значение

Период расчета, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             calc_period       // период расчета
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*calc\_period*

[in] Период расчета индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiWPR - [IND\\_WPR](#)).

## Группа технических индикаторов "Объемы"

Этот раздел содержит детали работы с группой классов технических индикаторов "Объемы" и описание соответствующих компонентов стандартной библиотеки MQL5.

Класс/группа	Описание
<a href="#"><u>CiAD</u></a>	Индикатор "Accumulation/Distribution"
<a href="#"><u>CiMFI</u></a>	Индикатор "Money Flow Index"
<a href="#"><u>CiOBV</u></a>	Индикатор "On Balance Volume"
<a href="#"><u>CiVolumes</u></a>	Индикатор "Volumes"

## Класс CiAD

Класс CiAD является классом для работы с техническим индикатором "Accumulation/Distribution".

### Описание

Класс CiAD обеспечивает создание, настройку и доступ к данным индикатора "Accumulation/Distribution".

### Декларация

```
class CiAD: public CIndicator
```

### Заголовок

```
#include <Indicators\Volumes.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiAD
```

### Методы класса по группам

Атрибуты	
<a href="#">Applied</a>	Получает период расчета
<a href="#">Создание</a>	
<a href="#">Create</a>	Создает индикатор
<a href="#">Доступ к данным</a>	
<a href="#">Main</a>	Получает данные буфера
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## Applied

Получает объект (тип объемов) применения.

```
ENUM_APPLIED_VOLUME Applied() const
```

### Возвращаемое значение

Объект (тип объемов) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    ENUM_APPLIED_VOLUME applied        // тип объемов
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*applied*

[in] Объект (тип объемов) применения индикатора ([ENUM\\_APPLIED\\_VOLUME](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiAD - [IND\\_AD](#)).

## Класс CiMFI

Класс CiMFI является классом для работы с техническим индикатором "Money Flow Index".

### Описание

Класс CiMFI обеспечивает создание, настройку и доступ к данным индикатора "Money Flow Index".

### Декларация

```
class CiMFI: public CIndicator
```

### Заголовок

```
#include <Indicators\Volumes.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiMFI
```

### Методы класса по группам

Атрибуты	
<a href="#">MaPeriod</a>	Получает период усреднения
<a href="#">Applied</a>	Получает объект (тип объемов) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## MaPeriod

Получает период усреднения.

```
int MaPeriod() const
```

### Возвращаемое значение

Период усреднения, назначенный при создании индикатора.

## Applied

Получает объект (тип объемов) применения.

```
ENUM_APPLIED_VOLUME Applied() const
```

### Возвращаемое значение

Объект (тип объемов) применения, назначенный при создании индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             ma_period,        // период усреднения
    ENUM_APPLIED_VOLUME applied        // тип объемов
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*ma\_period*

[in] Период усреднения индикатора.

*applied*

[in] Объект (тип объемов) применения индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiMFI - [IND\\_MFI](#)).

## Класс CiOBV

Класс CiOBV является классом для работы с техническим индикатором "On Balance Volume".

### Описание

Класс CiOBV обеспечивает создание, настройку и доступ к данным индикатора "On Balance Volume".

### Декларация

```
class CiOBV: public CIndicator
```

### Заголовок

```
#include <Indicators\Volumes.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
      CSeries
        CIndicator
          CiOBV
```

### Методы класса по группам

Атрибуты	
<a href="#">Applied</a>	Получает объект (тип объемов) применения
<a href="#">Создание</a>	
<a href="#">Create</a>	Создает индикатор
<a href="#">Доступ к данным</a>	
<a href="#">Main</a>	Получает данные буфера
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## Applied

Получает объект (тип объемов) применения.

```
ENUM_APPLIED_VOLUME Applied() const
```

### Возвращаемое значение

Объект (тип объемов) применения, назначенный при создании индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    ENUM_APPLIED_VOLUME applied        // тип объемов
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*applied*

[in] Объект (тип объемов) применения индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiOBV - [IND\\_OBV](#)).

## Класс CiVolumes

Класс CiVolumes является классом для работы с техническим индикатором "Volumes".

### Описание

Класс CiVolumes обеспечивает создание, настройку и доступ к данным индикатора "Volumes".

### Декларация

```
class CiVolumes: public CIndicator
```

### Заголовок

```
#include <Indicators\Volumes.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiVolumes
```

### Методы класса по группам

Атрибуты	
<a href="#">Applied</a>	Получает объект (тип объемов) применения
Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Main</a>	Получает данные буфера
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),  
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),  
[GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),  
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## Applied

Получает объект (тип объемов) применения.

```
ENUM_APPLIED_VOLUME Applied() const
```

### Возвращаемое значение

Объект (тип объемов) применения, назначенный при создании индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,      // символ
    ENUM_TIMEFRAMES period,     // период
    ENUM_APPLIED_VOLUME applied   // тип объемов
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение пересечения [ENUM\\_TIMEFRAMES](#)).

*applied*

[in] Объект (тип объемов) применения индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiVolumes - [IND\\_VOLUMES](#)).

## Группа технических индикаторов Билла Вильямса

Этот раздел содержит детали работы с группой классов технических индикаторов Билла Вильямса и описание соответствующих компонентов стандартной библиотеки MQL5.

Класс/группа	Описание
<a href="#"><u>CiAC</u></a>	Индикатор "Accelerator Oscillator"
<a href="#"><u>CiAlligator</u></a>	Индикатор "Alligator"
<a href="#"><u>CiAO</u></a>	Индикатор "Awesome Oscillator"
<a href="#"><u>CiFractals</u></a>	Индикатор "Fractals"
<a href="#"><u>CiGator</u></a>	Индикатор "Gator Oscillator"
<a href="#"><u>CiBWMFI</u></a>	Индикатор "Market Facilitation Index"

## Класс CiAC

Класс CiAC является классом для работы с техническим индикатором "Accelerator Oscillator".

### Описание

Класс CiAC обеспечивает создание, настройку и доступ к данным индикатора "Accelerator Oscillator".

### Декларация

```
class CiAC: public CIndicator
```

### Заголовок

```
#include <Indicators\BillWilliams.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiAC
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Main</a>	Получает данные буфера
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(  
    string          symbol,      // символ  
    ENUM_TIMEFRAMES period       // период  
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiAC - [IND\\_AC](#)).

## Класс CiAlligator

Класс CiAlligator является классом для работы с техническим индикатором "Alligator".

### Описание

Класс CiAlligator обеспечивает создание, настройку и доступ к данным индикатора "Alligator".

### Декларация

```
class CiAlligator: public CIndicator
```

### Заголовок

```
#include <Indicators\BillWilliams.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiAlligator
```

### Методы класса по группам

Атрибуты	
<a href="#">JawPeriod</a>	Получает период усреднения "челюстей"
<a href="#">JawShift</a>	Получает смещение "челюстей" по оси цен
<a href="#">TeethPeriod</a>	Получает период усреднения "зубов"
<a href="#">TeethShift</a>	Получает смещение "зубов" по оси цен
<a href="#">LipsPeriod</a>	Получает период усреднения "губ"
<a href="#">LipsShift</a>	Получает смещение "губ" по оси цен
<a href="#">MaMethod</a>	Получает метод усреднения
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Jaw</a>	Получает данные буфера линии "челюстей"
<a href="#">Teeth</a>	Получает данные буфера линии "зубов"

<a href="#">Lips</a>	Получает данные буфера линии "губ"
<b>Ввод/вывод</b>	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

**Методы унаследованные от CObject**Prev, Prev, Next, Next, [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)**Методы унаследованные от CArrayObj**[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)**Методы унаследованные от CSeries**[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)**Методы унаследованные от CIndicator**[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## JawPeriod

Получает период усреднения линии "челюстей".

```
int JawPeriod() const
```

### Возвращаемое значение

Период усреднения линии "челюстей", назначенный при создании индикатора.

## JawShift

Получает смещение линии "челюстей" по оси цен.

```
int JawShift() const
```

### Возвращаемое значение

Смещение линии "челюстей" по оси цен, назначенное при создании индикатора.

## TeethPeriod

Получает период усреднения линии "зубов".

```
int TeethPeriod() const
```

### Возвращаемое значение

Период усреднения линии "зубов", назначенный при создании индикатора.

## TeethShift

Получает смещение линии "зубов" по оси цен.

```
int TeethShift() const
```

### Возвращаемое значение

Смещение линии "зубов" по оси цен, назначенное при создании индикатора.

## LipsPeriod

Получает период усреднения линии "губ".

```
int LipsPeriod() const
```

### Возвращаемое значение

Период усреднения линии "губ", назначенный при создании индикатора.

## LipsShift

Получает смещение линии "губ" по оси цен.

```
int LipsShift() const
```

### Возвращаемое значение

Смещение линии "губ" по оси цен, назначенное при создании индикатора.

## MaMethod

Получает метод усреднения.

```
ENUM_MA_METHOD MaMethod() const
```

### Возвращаемое значение

Метод усреднения, назначенный при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             jaw_period,       // период "челюстей"
    int             jaw_shift,         // смещение "челюстей"
    int             teeth_period,      // период "зубов"
    int             teeth_shift,       // смещение "зубов"
    int             lips_period,       // период "губ"
    int             lips_shift,        // смещение "губ"
    ENUM_MA_METHOD ma_method,        // метод усреднения
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*jaw\_period*

[in] Период усреднения линии "челюстей".

*jaw\_shift*

[in] Смещение линии "челюстей" по оси цен.

*teeth\_period*

[in] Период усреднения линии "зубов".

*teeth\_shift*

[in] Смещение линии "зубов" по оси цен.

*lips\_period*

[in] Период усреднения линии "губ".

*lips\_shift*

[in] Смещение линии "губ" по оси цен.

*ma\_method*

[in] Метод усреднения индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Jaw

Получает элемент буфера линии "челюстей" по указанному индексу.

```
double Jaw(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии "челюстей".

### Возвращаемое значение

Элемент буфера линии "челюстей" по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Teeth

Получает элемент буфера линии "зубов" по указанному индексу.

```
double Teeth(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии "зубов".

### Возвращаемое значение

Элемент буфера линии "зубов" по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Lips

Получает элемент буфера линии "губ" по указанному индексу.

```
double Lips(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера линии "губ".

### Возвращаемое значение

Элемент буфера линии "губ" по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiAlligator - [IND\\_ALLIGATOR](#)).

## Класс CiAO

Класс CiAO является классом для работы с техническим индикатором "Awesome Oscillator".

### Описание

Класс CiAO обеспечивает создание, настройку и доступ к данным индикатора "Awesome Oscillator".

### Декларация

```
class CiAO: public CIndicator
```

### Заголовок

```
#include <Indicators\BillWilliams.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiAO
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Main</a>	Получает данные буфера
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(  
    string          symbol,      // символ  
    ENUM_TIMEFRAMES period       // период  
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiAO - [IND\\_AO](#)).

## Класс CiFractals

Класс CiFractals является классом для работы с техническим индикатором "Fractals".

### Описание

Класс CiFractals обеспечивает создание, настройку и доступ к данным индикатора "Fractals".

### Декларация

```
class CiFractals: public CIndicator
```

### Заголовок

```
#include <Indicators\BillWilliams.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiFractals
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Upper</a>	Получает данные верхнего буфера
<a href="#">Lower</a>	Получает данные нижнего буфера
Ввод/вывод	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

#### Методы унаследованные от CArrayObj

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#),

[CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#),  
[SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

#### Методы унаследованные от CSeries

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

#### Методы унаследованные от CIndicator

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#),  
[GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#),  
[DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(  
    string          symbol,      // символ  
    ENUM_TIMEFRAMES period       // период  
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось создать индикатор.

## Upper

Получает элемент верхнего буфера по указанному индексу.

```
double Upper(
    int index          // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента верхнего буфера.

### Возвращаемое значение

Элемент верхнего буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Lower

Получает элемент нижнего буфера по указанному индексу.

```
double Lower(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента нижнего буфера.

### Возвращаемое значение

Элемент нижнего буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiFractals - [IND\\_FRACTALS](#)).

## Класс CiGator

Класс CiGator является классом для работы с техническим индикатором "Gator oscillator".

### Описание

Класс CiGator обеспечивает создание, настройку и доступ к данным индикатора "Gator oscillator".

### Декларация

```
class CiGator: public CIndicator
```

### Заголовок

```
#include <Indicators\BillWilliams.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
    CIndicator
    CiGator
```

### Методы класса по группам

Атрибуты	
<a href="#">JawPeriod</a>	Получает период усреднения "челюстей"
<a href="#">JawShift</a>	Получает смещение "челюстей" по оси цен
<a href="#">TeethPeriod</a>	Получает период усреднения "зубов"
<a href="#">TeethShift</a>	Получает смещение "зубов" по оси цен
<a href="#">LipsPeriod</a>	Получает период усреднения "губ"
<a href="#">LipsShift</a>	Получает смещение "губ" по оси цен
<a href="#">MaMethod</a>	Получает метод усреднения
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
Создание	
<a href="#">Create</a>	Создает индикатор
Доступ к данным	
<a href="#">Upper</a>	Получает данные верхнего буфера
<a href="#">Lower</a>	Получает данные нижнего буфера

<b>Ввод/вывод</b>	
virtual <a href="#">Type</a>	Виртуальный метод идентификации

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)**Методы унаследованные от CArray**[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)**Методы унаследованные от CArrayObj**[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)**Методы унаследованные от CSeries**[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)**Методы унаследованные от CIndicator**[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## JawPeriod

Получает период усреднения линии "челюстей".

```
int JawPeriod() const
```

### Возвращаемое значение

Период усреднения линии "челюстей", назначенный при создании индикатора.

## JawShift

Получает смещение линии "челюстей" по оси цен.

```
int JawShift() const
```

### Возвращаемое значение

Смещение линии "челюстей" по оси цен, назначенное при создании индикатора.

## TeethPeriod

Получает период усреднения линии "зубов".

```
int TeethPeriod() const
```

### Возвращаемое значение

Период усреднения линии "зубов", назначенный при создании индикатора.

## TeethShift

Получает смещение линии "зубов" по оси цен.

```
int TeethShift() const
```

### Возвращаемое значение

Смещение линии "зубов" по оси цен, назначенное при создании индикатора.

## LipsPeriod

Получает период усреднения линии "губ".

```
int LipsPeriod() const
```

### Возвращаемое значение

Период усреднения линии "губ", назначенный при создании индикатора.

## LipsShift

Получает смещение линии "губ" по оси цен.

```
int LipsShift() const
```

### Возвращаемое значение

Смещение линии "губ" по оси цен, назначенное при создании индикатора.

## MaMethod

Получает метод усреднения.

```
ENUM_MA_METHOD MaMethod() const
```

### Возвращаемое значение

Метод усреднения, назначенный при создании индикатора.

## Applied

Получает объект (тип цены, хэндл) применения.

```
int Applied() const
```

### Возвращаемое значение

Объект (тип цены, хэндл) применения, назначенный при создании индикатора.

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    int             jaw_period,       // период "челюстей"
    int             jaw_shift,         // смещение "челюстей"
    int             teeth_period,      // период "зубов"
    int             teeth_shift,       // смещение "зубов"
    int             lips_period,       // период "губ"
    int             lips_shift,        // смещение "губ"
    ENUM_MA_METHOD ma_method,        // метод усреднения
    int             applied           // тип цены, хэндл
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*jaw\_period*

[in] Период усреднения линии "челюстей".

*jaw\_shift*

[in] Смещение линии "челюстей" по оси цен.

*teeth\_period*

[in] Период усреднения линии "зубов".

*teeth\_shift*

[in] Смещение линии "зубов" по оси цен.

*lips\_period*

[in] Период усреднения линии "губ".

*lips\_shift*

[in] Смещение линии "губ" по оси цен.

*ma\_method*

[in] Метод усреднения индикатора (значение перечисления [ENUM\\_MA\\_METHOD](#)).

*applied*

[in] Объект (тип цены, хэндл) применения индикатора.

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.



## Upper

Получает элемент верхнего буфера по указанному индексу.

```
double Upper(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента верхнего буфера.

### Возвращаемое значение

Элемент верхнего буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Lower

Получает элемент нижнего буфера по указанному индексу.

```
double Lower(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента нижнего буфера.

### Возвращаемое значение

Элемент нижнего буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiGator - [IND\\_GATOR](#)).

## Класс CiBWMFI

Класс CiBWMFI является классом для работы с техническим индикатором "Market Facilitation Index by Bill Williams".

### Описание

Класс CiBWMFI обеспечивает создание, настройку и доступ к данным индикатора "Market Facilitation Index by Bill Williams".

### Декларация

```
class CiBWMFI: public CIndicator
```

### Заголовок

```
#include <Indicators\BillWilliams.mqh>
```

### Иерархия наследования

```
CObject
  CArray
    CArrayObj
    CSeries
      CIndicator
        CiBWMFI
```

### Методы класса по группам

Атрибуты	
<a href="#">Applied</a>	Получает объект (тип цены, хэндл) применения
<b>Создание</b>	
<a href="#">Create</a>	Создает индикатор
<b>Доступ к данным</b>	
<a href="#">Main</a>	Получает данные буфера
<b>Ввод/вывод</b>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Compare](#)

#### Методы унаследованные от CArray

[Step](#), [Step](#), [Total](#), [Available](#), [Max](#), [IsSorted](#), [SortMode](#), [Clear](#), [Sort](#)

**Методы унаследованные от CArrayObj**

[FreeMode](#), [FreeMode](#), [Save](#), [Load](#), [CreateElement](#), [Reserve](#), [Resize](#), [Shutdown](#), [Add](#), [AddArray](#), [Insert](#), [InsertArray](#), [AssignArray](#), [At](#), [Update](#), [Shift](#), [Detach](#), [Delete](#), [DeleteRange](#), [Clear](#), [CompareArray](#), [InsertSort](#), [Search](#), [SearchGreat](#), [SearchLess](#), [SearchGreatOrEqual](#), [SearchLessOrEqual](#), [SearchFirst](#), [SearchLast](#)

**Методы унаследованные от CSeries**

[Name](#), [BuffersTotal](#), [BufferSize](#), [Timeframe](#), [Symbol](#), [Period](#), [PeriodDescription](#), [RefreshCurrent](#)

**Методы унаследованные от CIndicator**

[Handle](#), [Status](#), [FullRelease](#), [Redrawer](#), [Create](#), [BufferResize](#), [BarsCalculated](#), [GetData](#), [GetData](#), [GetData](#), [Minimum](#), [MinValue](#), [Maximum](#), [MaxValue](#), [Refresh](#), [AddToChart](#), [DeleteFromChart](#), [MethodDescription](#), [PriceDescription](#), [VolumeDescription](#)

## Applied

Получает объект (тип объемов) применения.

```
ENUM_APPLIED_VOLUME Applied() const
```

### Возвращаемое значение

Объект (тип объемов) применения, назначенный при создании индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

## Create

Создает индикатор с указанными параметрами. Для обновления и получения значений индикатора используйте [Refresh\(\)](#) и [GetData\(\)](#).

```
bool Create(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    ENUM_APPLIED_VOLUME applied        // тип объемов
)
```

### Параметры

*symbol*

[in] Рабочий символ индикатора.

*period*

[in] Рабочий период индикатора (значение перечисления [ENUM\\_TIMEFRAMES](#)).

*applied*

[in] Объект (тип объемов) применения индикатора (значение перечисления [ENUM\\_APPLIED\\_VOLUME](#)).

### Возвращаемое значение

true - в случае удачи, false - если не удалось создать индикатор.

## Main

Получает элемент буфера по указанному индексу.

```
double Main(
    int index           // индекс
) const
```

### Параметры

*index*

[in] Индекс элемента буфера.

### Возвращаемое значение

Элемент буфера по указанному индексу, либо [EMPTY\\_VALUE](#) если нет корректных данных.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiBWMFI - [IND\\_BWMFI](#)).

## Класс CiCustom

Класс CiCustom является классом для работы с техническим индикатором пользовательского типа.

### Описание

Класс CiCustom обеспечивает создание, настройку и доступ к данным индикатора пользовательского типа.

### Декларация

```
class CiCustom: public CIndicator
```

### Заголовок

```
#include <Indicators\Custom.mqh>
```

### Методы класса по группам

Атрибуты	
<a href="#">NumBuffers</a>	Устанавливает количество буферов индикатора
<a href="#">NumParams</a>	Получает количество параметров использованных при создании индикатора
<a href="#">ParamType</a>	Получает тип указанного параметра
<a href="#">ParamLong</a>	Получает значение указанного целочисленного параметра
<a href="#">ParamDouble</a>	Получает значение указанного параметра с плавающей точкой
<a href="#">ParamString</a>	Получает значение указанного строкового параметра
<a href="#">Ввод/вывод</a>	
<a href="#">virtual Type</a>	Виртуальный метод идентификации

## NumBuffers

Устанавливает количество буферов индикатора.

```
bool NumBuffers()
```

### Возвращаемое значение

`true` - в случае удачи, `false` - если не удалось установить нужное количество буферов.

## NumParams

Получает количество параметров использованных при создании индикатора.

```
int NumParams() const
```

### Возвращаемое значение

Количество параметров использованных при создании индикатора.

## ParamType

Получает тип указанного параметра.

```
ENUM_DATATYPE ParamType (
    int index      // номер
) const
```

### Параметры

*index*

[in] Номер параметра.

### Возвращаемое значение

Получает тип указанного параметра, использованного при создании индикатора.

### Примечание

Если номер параметра указан неверно, вернётся [WRONG\\_VALUE](#).

## ParamLong

Получает значение указанного целочисленного параметра.

```
long ParamLong(
    int index          // номер
) const
```

### Параметры

*index*

[in] Номер параметра.

### Возвращаемое значение

Значение указанного целочисленного параметра, использованного при создании индикатора.

### Примечание

Если номер параметра указан неверно или тип параметра не целочисленный, вернётся 0.

## ParamDouble

Получает значение указанного параметра с плавающей точкой.

```
double ParamDouble(
    int index           // номер
) const
```

### Параметры

*index*

[in] Номер параметра.

### Возвращаемое значение

Значение указанного параметра с плавающей точкой, использованного при создании индикатора.

### Примечание

Если номер параметра указан неверно или тип параметра не с плавающей точкой, вернется [EMPTY\\_VALUE](#).

## ParamString

Получает значение указанного строкового параметра.

```
string ParamString(  
    int index // номер  
) const
```

### Параметры

*index*

[in] Номер параметра.

### Возвращаемое значение

Значение указанного строкового параметра, использованного при создании индикатора.

### Примечание

Если номер параметра указан неверно или тип параметра не строковый, вернется пустая строка.

## Type

Виртуальный метод идентификации.

```
virtual int Type() const
```

### Возвращаемое значение

Тип индикатора (для CiCustom - [IND\\_CUSTOM](#)).

## Торговые классы

Этот раздел содержит технические детали работы с торговыми классами и описание соответствующих компонентов стандартной библиотеки MQL5.

Использование торговых классов позволит сэкономить время при создании пользовательских программ (экспертов).

Стандартная библиотека MQL5 (в части торговых классов) размещается в рабочем каталоге терминала в папке `Include\Trade`.

Класс/группа	Описание
<a href="#"><u>CAccountInfo</u></a>	Класс для работы со свойствами торгового счета
<a href="#"><u>CSymbolInfo</u></a>	Класс для работы со свойствами торгового инструмента
<a href="#"><u>COrderInfo</u></a>	Класс для работы со свойствами отложенного ордера
<a href="#"><u>CHistoryOrderInfo</u></a>	Класс для работы со свойствами "исторического" ордера
<a href="#"><u>CPositionInfo</u></a>	Класс для работы со свойствами открытой позиции
<a href="#"><u>CDealInfo</u></a>	Класс для работы со свойствами "исторической" сделки
<a href="#"><u>CTrade</u></a>	Класс для совершения торговых операций
<a href="#"><u>CTerminalInfo</u></a>	Класс для доступа к параметрам окружения терминала

## Класс CAccountInfo

Класс CAccountInfo является классом для упрощенного доступа к свойствам текущего открытого торгового счета.

### Описание

Класс CAccountInfo обеспечивает доступ к свойствам текущего открытого торгового счета.

### Декларация

```
class CAccountInfo : public CObject
```

### Заголовок

```
#include <Trade\AccountInfo.mqh>
```

### Иерархия наследования

CObject

CAccountInfo

### Методы класса по группам

Доступ к целочисленным свойствам	
<a href="#"><u>Login</u></a>	Получает номер счета
<a href="#"><u>TradeMode</u></a>	Получает режим торговли
<a href="#"><u>TradeModeDescription</u></a>	Получает режим торговли как строку
<a href="#"><u>Leverage</u></a>	Получает размер предоставленного кредитного плеча
<a href="#"><u>StopoutMode</u></a>	Получает режим задания минимального уровня залога
<a href="#"><u>StopoutModeDescription</u></a>	Получает режим задания минимального уровня залога как строку
<a href="#"><u>TradeAllowed</u></a>	Получает флаг разрешения торговли
<a href="#"><u>TradeExpert</u></a>	Получает флаг разрешения автоматизированной торговли
<a href="#"><u>LimitOrders</u></a>	Получает максимально допустимое количество действующих отложенных ордеров
<a href="#"><u>MarginMode</u></a>	Получает режим расчета маржи
<a href="#"><u>MarginModeDescription</u></a>	Получает режим расчета маржи как строку
<b>Доступ к double-свойствам</b>	

<a href="#"><u>Balance</u></a>	Получает баланс счета
<a href="#"><u>Credit</u></a>	Получает размер предоставленного кредита
<a href="#"><u>Profit</u></a>	Получает значение текущей прибыли по счету
<a href="#"><u>Equity</u></a>	Получает значение собственных средств на счете
<a href="#"><u>Margin</u></a>	Получает размер зарезервированных залоговых средств
<a href="#"><u>FreeMargin</u></a>	Получает размер свободных средств
<a href="#"><u>MarginLevel</u></a>	Получает уровень залоговых средств
<a href="#"><u>MarginCall</u></a>	Получает уровень залоговых средств пополнения счета
<a href="#"><u>MarginStopOut</u></a>	Получает уровень залоговых средств принудительного закрытия
<b>Доступ к текстовым свойствам</b>	
<a href="#"><u>Name</u></a>	Получает имя клиента
<a href="#"><u>Server</u></a>	Получает наименование торгового сервера
<a href="#"><u>Currency</u></a>	Получает наименование валюты депозита
<a href="#"><u>Company</u></a>	Получает наименование компании, обслуживающей счет
<b>Доступ к функциям API MQL5</b>	
<a href="#"><u>InfoInteger</u></a>	Получает значение указанного целочисленного свойства счета
<a href="#"><u>InfoDouble</u></a>	Получает значение указанного double-свойства счета
<a href="#"><u>InfoString</u></a>	Получает значение указанного текстового свойства счета
<b>Дополнительные методы</b>	
<a href="#"><u>OrderProfitCheck</u></a>	Вычисляет размер прибыли на основании переданных параметров
<a href="#"><u>MarginCheck</u></a>	Получает размер маржи, необходимой для торговой операции
<a href="#"><u>FreeMarginCheck</u></a>	Получает размер свободной маржи, которая останется после торговой операции
<a href="#"><u>MaxLotCheck</u></a>	Получает максимально возможный объем торговой операции

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## Login

Получает номер счета.

```
long Login() const
```

### Возвращаемое значение

Номер счета.

## TradeMode

Получает режим торговли.

```
ENUM_ACCOUNT_TRADE_MODE TradeMode() const
```

### Возвращаемое значение

Режим торговли из перечисления [ENUM\\_ACCOUNT\\_TRADE\\_MODE](#).

## TradeModeDescription

Получает режим торговли как строку.

```
string TradeModeDescription() const
```

### Возвращаемое значение

Режим торговли как строка.

## Leverage

Получает размер предоставленного кредитного плеча.

```
long Leverage() const
```

### Возвращаемое значение

Размер предоставленного кредитного плеча.

## StopoutMode

Получает режим задания минимального уровня залога.

```
ENUM_ACCOUNT_STOPOUT_MODE StopoutMode() const
```

### Возвращаемое значение

Режим задания минимального уровня залога из перечисления [ENUM\\_ACCOUNT\\_STOPOUT\\_MODE](#).

## StopoutModeDescription

Получает режим задания минимального уровня залога как строку.

```
string StopoutModeDescription() const
```

### Возвращаемое значение

Режим задания минимального уровня залога как строка.

## MarginMode

Получает режим расчета маржи.

```
ENUM_ACCOUNT_MARGIN_MODE MarginMode() const
```

### Возвращаемое значение

Режим расчета маржи из перечисления [ENUM\\_ACCOUNT\\_MARGIN\\_MODE](#).

## MarginModeDescription

Получает режим расчета маржи как строку.

```
string MarginModeDescription() const
```

### Возвращаемое значение

Режим расчета маржи в виде строки.

## TradeAllowed

Получает флаг разрешения торговли.

```
bool TradeAllowed() const
```

### Возвращаемое значение

Флаг разрешения торговли.

## TradeExpert

Получает флаг разрешения автоматизированной торговли.

```
bool TradeExpert() const
```

### Возвращаемое значение

Флаг разрешения автоматизированной торговли.

## LimitOrders

Получает максимально допустимое количество действующих отложенных ордеров.

```
int LimitOrders() const
```

### Возвращаемое значение

Максимально допустимое количество действующих отложенных ордеров.

### Примечание

0 - ограничений нет.

## Balance

Получает баланс счета.

```
double Balance() const
```

### Возвращаемое значение

Баланс счета в валюте депозита.

## Credit

Получает размер предоставленного кредита.

```
double Credit() const
```

### Возвращаемое значение

Размер предоставленного кредита в валюте депозита.

## Profit

Получает значение текущей прибыли по счету.

```
double Profit() const
```

### Возвращаемое значение

Значение текущей прибыли по счету в валюте депозита.

## Equity

Получает значение собственных средств на счете.

```
double Equity() const
```

### Возвращаемое значение

Значение собственных средств на счете в валюте депозита.

## Margin

Получает размер зарезервированных залоговых средств.

```
double Margin() const
```

### Возвращаемое значение

Размер зарезервированных залоговых средств в валюте депозита.

## FreeMargin

Получает размер свободных средств.

```
double FreeMargin() const
```

### Возвращаемое значение

Размер свободных средств в валюте депозита.

## MarginLevel

Получает уровень залоговых средств.

```
double MarginLevel() const
```

### Возвращаемое значение

Уровень залоговых средств.

## MarginCall

Получает уровень залоговых средств пополнения счета.

```
double MarginCall() const
```

### Возвращаемое значение

Уровень залоговых средств пополнения счета.

## MarginStopOut

Получает уровень залоговых средств принудительного закрытия.

```
double MarginStopOut() const
```

### Возвращаемое значение

Уровень залоговых средств принудительного закрытия.

## Name

Получает имя клиента.

```
string Name() const
```

### Возвращаемое значение

Имя клиента.

## Server

Получает наименование торгового сервера.

```
string Server() const
```

### Возвращаемое значение

Наименование торгового сервера.

## Currency

Получает наименование валюты депозита.

```
string Currency() const
```

### Возвращаемое значение

Наименование валюты депозита.

## Company

Получает наименование компании, обслуживающей счет.

```
string Company() const
```

### Возвращаемое значение

Наименование компании, обслуживающей счет.

## InfolInteger

Получает значение указанного целочисленного свойства счета.

```
long InfoInteger(
    ENUM_ACCOUNT_INFO_INTEGER prop_id      // идентификатор свойства
) const
```

### Параметры

*prop\_id*

[in] Идентификатор свойства. Значение может быть одним из значений [ENUM\\_ACCOUNT\\_INFO\\_INTEGER](#).

### Возвращаемое значение

Значение типа [long](#).

## InfoDouble

Получает значение указанного double-свойства счета.

```
double InfoDouble(  
    ENUM_ACCOUNT_INFO_DOUBLE prop_id // идентификатор свойства  
) const
```

### Параметры

*prop\_id*

[in] Идентификатор свойства. Значение может быть одним из значений [ENUM\\_ACCOUNT\\_INFO\\_DOUBLE](#).

### Возвращаемое значение

Значение типа [double](#).

## InfoString

Получает значение указанного текстового свойства счета.

```
string InfoString(  
    ENUM_ACCOUNT_INFO_STRING prop_id      // идентификатор свойства  
) const
```

### Параметры

*prop\_id*

[in] Идентификатор свойства. Значение может быть одним из значений [ENUM\\_ACCOUNT\\_INFO\\_STRING](#).

### Возвращаемое значение

Значение типа [string](#).

## OrderProfitCheck

Вычисляет размер прибыли для текущего счета на основании переданных параметров. Предназначена для предварительной оценки результата торговой операции. Значение возвращается в валюте счета.

```
double OrderProfitCheck(
    const string      symbol,           // символ
    ENUM_ORDER_TYPE   trade_operation,   // тип ордера (ORDER_TYPE_BUY или ORDER_TYPE_SELL)
    double            volume,            // объем
    double            price_open,         // цена открытия позиции
    double            price_close        // цена закрытия позиции
) const
```

### Параметры

*symbol*

[in] Символ, по которому предполагается совершить торговую операцию.

*trade\_operation*

[in] Тип торговой операции из перечисления [ENUM\\_ORDER\\_TYPE](#).

*volume*

[in] Объем торговой операции.

*price\_open*

[in] Цена открытия.

*price\_close*

[in] Цена закрытия.

### Возвращаемое значение

Возвращает размер прибыли в случае успеха, или [EMPTY\\_VALUE](#) в случае ошибки.

## MarginCheck

Получает размер маржи, необходимой для торговой операции.

```
double MarginCheck(
    const string      symbol,           // символ
    ENUM_ORDER_TYPE   trade_operation,  // тип ордера (ORDER_TYPE_BUY или ORDER_TYPE_SELL)
    double            volume,           // объем
    double            price             // цена
) const
```

### Параметры

*symbol*

[in] Символ, по которому предполагается совершить торговую операцию.

*trade\_operation*

[in] Тип торговой операции из перечисления [ENUM\\_ORDER\\_TYPE](#).

*volume*

[in] Объем торговой операции.

*price*

[in] Цена торговой операции.

### Возвращаемое значение

Размер маржи, необходимой для торговой операции.

## FreeMarginCheck

Получает размер свободной маржи, которая останется после торговой операции.

```
double FreeMarginCheck(
    const string      symbol,           // символ
    ENUM_ORDER_TYPE   trade_operation,  // тип ордера (ORDER_TYPE_BUY или ORDER_TYPE_SELL)
    double            volume,           // объем
    double            price,            // цена
) const
```

### Параметры

*symbol*

[in] Символ, по которому предполагается совершить торговую операцию.

*trade\_operation*

[in] Тип торговой операции из перечисления [ENUM\\_ORDER\\_TYPE](#).

*volume*

[in] Объем торговой операции.

*price*

[in] Цена торговой операции.

### Возвращаемое значение

Размер свободной маржи, которая останется после торговой операции.

## MaxLotCheck

Получает максимально возможный объем торговой операции.

```
double MaxLotCheck(
    const string      symbol,           // символ
    ENUM_ORDER_TYPE   trade_operation,  // тип ордера (ORDER_TYPE_BUY или ORDER_TYPE_SELL)
    double            price,            // цена
    double            percent=100        // доля свободной маржи (по умолчанию 100%)
) const
```

### Параметры

*symbol*

[in] Символ, по которому предполагается совершить торговую операцию.

*trade\_operation*

[in] Тип торговой операции из перечисления [ENUM\\_ORDER\\_TYPE](#).

*price*

[in] Цена торговой операции.

*percent=100*

[in] Доля свободной маржи (в процентах), которую предполагается использовать для осуществления торговой операции.

### Возвращаемое значение

Максимально возможный объем торговой операции.

## Класс CSymbolInfo

Класс CSymbolInfo является классом для упрощенного доступа к свойствам символа.

### Описание

Класс CSymbolInfo обеспечивает доступ к свойствам символа.

### Декларация

```
class CSymbolInfo : public CObject
```

### Заголовок

```
#include <Trade\SymbolInfo.mqh>
```

### Иерархия наследования

[CObject](#)

CSymbolInfo

### Методы класса по группам

Управление	
<a href="#">Refresh</a>	Обновляет данные по символу
<a href="#">RefreshRates</a>	Обновляет котировки по символу
Свойства	
<a href="#">Name</a>	Получает/устанавливает наименование финансового инструмента
<a href="#">Select</a>	Получает/устанавливает флаг символа "Символ в Обзоре Рынка"
<a href="#">IsSynchronized</a>	Проверяет синхронизацию символа с сервером
Объемы	
<a href="#">Volume</a>	Получает объем последней сделки
<a href="#">VolumeHigh</a>	Получает максимальный объем за день
<a href="#">VolumeLow</a>	Получает минимальный объем за день
Разное	
<a href="#">Time</a>	Получает время последней котировки
<a href="#">Spread</a>	Получает размер спреда в пунктах
<a href="#">SpreadFloat</a>	Получает признак плавающего спреда
<a href="#">TicksBookDepth</a>	Получает глубину хранения тиков

<b>Уровни</b>	
<a href="#"><u>StopsLevel</u></a>	Получает минимальный отступ для ордеров в пунктах
<a href="#"><u>FreezeLevel</u></a>	Получает дистанцию заморозки торговых операций в пунктах
<b>Цены спроса</b>	
<a href="#"><u>Bid</u></a>	Получает текущую цену Bid
<a href="#"><u>BidHigh</u></a>	Получает максимальную цену Bid за день
<a href="#"><u>BidLow</u></a>	Получает минимальную цену Bid за день
<b>Цены предложения</b>	
<a href="#"><u>Ask</u></a>	Получает текущую цену Ask
<a href="#"><u>AskHigh</u></a>	Получает максимальную цену Ask за день
<a href="#"><u>AskLow</u></a>	Получает минимальную цену Ask за день
<b>Цены</b>	
<a href="#"><u>Last</u></a>	Получает текущую цену Last
<a href="#"><u>LastHigh</u></a>	Получает максимальную цену Last за день
<a href="#"><u>LastLow</u></a>	Получает минимальную цену Last за день
<b>Режимы торговли</b>	
<a href="#"><u>TradeCalcMode</u></a>	Получает способ вычисления стоимости контрактов
<a href="#"><u>TradeCalcModeDescription</u></a>	Получает способ вычисления стоимости контрактов как строку
<a href="#"><u>TradeMode</u></a>	Получает тип исполнения ордеров
<a href="#"><u>TradeModeDescription</u></a>	Получает тип исполнения ордеров как строку
<a href="#"><u>TradeExecution</u></a>	Получает режим заключения сделок
<a href="#"><u>TradeExecutionDescription</u></a>	Получает режим заключения сделок как строку
<b>Свопы</b>	
<a href="#"><u>SwapMode</u></a>	Получает модель расчета свопа
<a href="#"><u>SwapModeDescription</u></a>	Получает модель расчета свопа как строку
<a href="#"><u>SwapRollover3days</u></a>	Получает день недели начисления тройного свопа
<a href="#"><u>SwapRollover3daysDescription</u></a>	Получает день недели начисления тройного свопа как строку

<b>Залоги и флаги</b>	
<a href="#"><u>MarginInitial</u></a>	Получает значение начальной маржи
<a href="#"><u>MarginMaintenance</u></a>	Получает значение поддерживающей маржи
<a href="#"><u>MarginLong</u></a>	Получает коэффициент взимания маржи по длинным позициям
<a href="#"><u>MarginShort</u></a>	Получает коэффициент взимания маржи по коротким позициям
<a href="#"><u>MarginLimit</u></a>	Получает коэффициент взимания маржи по Limit ордерам
<a href="#"><u>MarginStop</u></a>	Получает коэффициент взимания маржи по Stop ордерам
<a href="#"><u>MarginStopLimit</u></a>	Получает коэффициент взимания маржи по Stop Limit ордерам
<a href="#"><u>TradeTimeFlags</u></a>	Получает флаги разрешённых режимов истечения ордера
<a href="#"><u>TradeFillFlags</u></a>	Получает флаги разрешённых режимов заливки ордера
<b>Квантование</b>	
<a href="#"><u>Digits</u></a>	Получает количество знаков после десятичной точки
<a href="#"><u>Point</u></a>	Получает значение одного пункта
<a href="#"><u>TickValue</u></a>	Получает стоимость минимального изменения цены
<a href="#"><u>TickValueProfit</u></a>	Получает рассчитанную стоимость тика для прибыльной позиции
<a href="#"><u>TickValueLoss</u></a>	Получает рассчитанную стоимость тика для убыточной позиции
<a href="#"><u>TickSize</u></a>	Получает минимальное изменение цены
<b>Размеры контрактов</b>	
<a href="#"><u>ContractSize</u></a>	Получает размер торгового контракта
<a href="#"><u>LotsMin</u></a>	Получает минимальный объем для заключения сделки
<a href="#"><u>LotsMax</u></a>	Получает максимальный объем для заключения сделки
<a href="#"><u>LotsStep</u></a>	Получает минимальный шаг изменения объема для заключения сделки

<a href="#"><u>LotsLimit</u></a>	Получает максимально допустимый совокупный объем открытой позиции и отложенных ордеров на символе
<b>Размеры свопов</b>	
<a href="#"><u>SwapLong</u></a>	Получает значение свопа длинной позиции
<a href="#"><u>SwapShort</u></a>	Получает значение свопа короткой позиции
<b>Текстовые свойства</b>	
<a href="#"><u>CurrencyBase</u></a>	Получает наименование базовой валюты символа
<a href="#"><u>CurrencyProfit</u></a>	Получает наименование валюты прибыли
<a href="#"><u>CurrencyMargin</u></a>	Получает наименование валюты залога
<a href="#"><u>Bank</u></a>	Получает наименование источника текущей котировки
<a href="#"><u>Description</u></a>	Получает строковое описание символа
<a href="#"><u>Path</u></a>	Получает путь в дереве символов
<b>Свойства символов</b>	
<a href="#"><u>SessionDeals</u></a>	Получает количество сделок в текущей сессии
<a href="#"><u>SessionBuyOrders</u></a>	Получает общее число ордеров на покупку в текущий момент
<a href="#"><u>SessionSellOrders</u></a>	Получает общее число ордеров на продажу в текущий момент
<a href="#"><u>SessionTurnover</u></a>	Получает суммарный оборот в текущую сессию
<a href="#"><u>SessionInterest</u></a>	Получает суммарный объем открытых позиций
<a href="#"><u>SessionBuyOrdersVolume</u></a>	Получает общий объем ордеров на покупку в текущий момент
<a href="#"><u>SessionSellOrdersVolume</u></a>	Получает общий объем ордеров на продажу в текущий момент
<a href="#"><u>SessionOpen</u></a>	Получает цену открытия текущей сессии
<a href="#"><u>SessionClose</u></a>	Получает цену закрытия текущей сессии
<a href="#"><u>SessionAW</u></a>	Получает средневзвешенную цену текущей сессии
<a href="#"><u>SessionPriceSettlement</u></a>	Получает цену поставки на текущую сессию
<a href="#"><u>SessionPriceLimitMin</u></a>	Получает минимально допустимое значение цены за текущую сессию

<a href="#">SessionPriceLimitMax</a>	Получает максимально допустимое значение цены за текущую сессию
<a href="#">Доступ к функциям API MQL5</a>	
<a href="#">InfoInteger</a>	Получает значение указанного целочисленного свойства
<a href="#">InfoDouble</a>	Получает значение указанного double-свойства
<a href="#">InfoString</a>	Получить значение указанного текстового свойства
<a href="#">Сервисные функции</a>	
<a href="#">NormalizePrice</a>	Нормализует цену с учетом свойств символа

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## Refresh

Обновляет данные по символу.

```
void Refresh()
```

### Возвращаемое значение

Нет.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## RefreshRates

Обновляет котировки по символу.

```
bool RefreshRates()
```

### Возвращаемое значение

true - в случае удачи, false - если не удалось обновить котировки.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Name

Получает наименование финансового инструмента.

```
string Name() const
```

### Возвращаемое значение

Наименование финансового инструмента.

## Name

Устанавливает наименование финансового инструмента для дальнейшей работы с ним.

```
bool Name(string name)
```

### Возвращаемое значение

Нет.

## Select

Получает флаг символа "Символ в обзоре рынка".

```
bool Select() const
```

### Возвращаемое значение

Флаг символа "Символ в обзоре рынка".

## Select

Устанавливает флаг символа "Символ в обзоре рынка".

```
bool Select()
```

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить флаг.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## IsSynchronized

Проверяет синхронизацию символа с сервером.

```
bool IsSynchronized() const
```

### Возвращаемое значение

true - если символ синхронизирован с сервером, false - если нет.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Volume

Получает объем последней сделки.

```
long Volume() const
```

### Возвращаемое значение

Объем последней сделки.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## VolumeHigh

Получает максимальный объем за день.

```
long VolumeHigh() const
```

### Возвращаемое значение

Максимальный объем за день.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## VolumeLow

Получает минимальный объем за день.

```
long VolumeLow() const
```

### Возвращаемое значение

Минимальный объем за день.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Time

Получает время последней котировки.

```
datetime Time() const
```

### Возвращаемое значение

Время последней котировки.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Spread

Получает размер спреда в пунктах.

```
int Spread() const
```

### Возвращаемое значение

Размер спреда в пунктах.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SpreadFloat

Получает признак плавающего спреда.

```
bool SpreadFloat() const
```

### Возвращаемое значение

Признак плавающего спреда.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TicksBookDepth

Получает глубину хранения тиков.

```
int TicksBookDepth() const
```

### Возвращаемое значение

Глубина хранения тиков.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## **StopsLevel**

Получает минимальный отступ для ордеров в пунктах.

```
int StopsLevel() const
```

### **Возвращаемое значение**

Минимальный отступ для ордеров в пунктах.

### **Примечание**

Символ должен быть предварительно выбран методом [Name](#).

## FreezeLevel

Получает дистанцию заморозки торговых операций в пунктах.

```
int  FreezeLevel() const
```

### Возвращаемое значение

Дистанция заморозки торговых операций в пунктах.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Bid

Получает текущую цену Bid.

```
double Bid() const
```

### Возвращаемое значение

Текущая цена Bid.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## BidHigh

Получает максимальную цену Bid за день.

```
double BidHigh() const
```

### Возвращаемое значение

Максимальная цена Bid за день.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## BidLow

Получает минимальную цену Bid за день.

```
double BidLow() const
```

### Возвращаемое значение

Минимальная цена Bid за день.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Ask

Получает текущую цену Ask.

```
double Ask() const
```

### Возвращаемое значение

Текущая цена Ask.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## AskHigh

Получает максимальную цену Ask за день.

```
double AskHigh() const
```

### Возвращаемое значение

Максимальная цена Ask за день.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## AskLow

Получает минимальную цену Ask за день.

```
double AskLow() const
```

### Возвращаемое значение

Минимальная цена Ask за день.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Last

Получает текущую цену Last.

```
double Last() const
```

### Возвращаемое значение

Текущая цена Last.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## LastHigh

Получает максимальную цену Last за день.

```
double LastHigh() const
```

### Возвращаемое значение

Максимальная цена Last за день.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## LastLow

Получает минимальную цену Last за день.

```
double LastLow() const
```

### Возвращаемое значение

Минимальная цена Last за день.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TradeCalcMode

Получает способ вычисления стоимости контрактов.

```
ENUM_SYMBOL_CALC_MODE TradeCalcMode() const
```

### Возвращаемое значение

Способ вычисления стоимости контрактов из перечисления [ENUM\\_SYMBOL\\_CALC\\_MODE](#).

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TradeCalcModeDescription

Получает способ вычисления стоимости контрактов как строку.

```
string TradeCalcModeDescription() const
```

### Возвращаемое значение

Способ вычисления стоимости контрактов как строка.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TradeMode

Получает тип исполнения ордеров.

```
ENUM_SYMBOL_TRADE_MODE TradeMode() const
```

### Возвращаемое значение

Тип исполнения ордеров из перечисления [ENUM\\_SYMBOL\\_TRADE\\_MODE](#).

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TradeModeDescription

Получает тип исполнения ордеров как строку.

```
string TradeModeDescription() const
```

### Возвращаемое значение

Тип исполнения ордеров как строка.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TradeExecution

Получает режим заключения сделок.

```
ENUM_SYMBOL_TRADE_EXECUTION TradeExecution() const
```

### Возвращаемое значение

Режим заключения сделок из перечисления [ENUM\\_SYMBOL\\_TRADE\\_EXECUTION](#).

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TradeExecutionDescription

Получает режим заключения сделок как строку.

```
string TradeExecutionDescription() const
```

### Возвращаемое значение

Режим заключения сделок как строка.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SwapMode

Получает модель расчета свопа.

```
ENUM_SYMBOL_SWAP_MODE SwapMode() const
```

### Возвращаемое значение

Модель расчета свопа из перечисления [ENUM\\_SYMBOL\\_SWAP\\_MODE](#).

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SwapModeDescription

Получает модель расчета свопа как строку.

```
string SwapModeDescription() const
```

### Возвращаемое значение

Модель расчета свопа как строка.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SwapRollover3days

Получает день недели начисления тройного свопа.

```
ENUM_DAY_OF_WEEK SwapRollover3days() const
```

### Возвращаемое значение

День недели начисления тройного свопа из перечисления [ENUM\\_DAY\\_OF\\_WEEK](#).

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SwapRollover3daysDescription

Получает день недели начисления тройного свопа как строку.

```
string SwapRollover3daysDescription() const
```

### Возвращаемое значение

День недели начисления тройного свопа как строка.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## MarginInitial

Получает значение начальной маржи.

```
double MarginInitial()
```

### Возвращаемое значение

Значение начальной маржи.

### Примечание

Указывает размер маржи в маржинальной валюте инструмента, взимаемую с одного лота.  
Используется при проверке средств клиента при входе в рынок.

Символ должен быть предварительно выбран методом [Name](#).

## MarginMaintenance

Получает значение поддерживающей маржи.

```
double MarginMaintenance()
```

### Возвращаемое значение

Значение поддерживающей маржи.

### Примечание

Указывает размер маржи в маржинальной валюте инструмента, взимаемой с одного лота. Используется для проверки средств клиента, при изменении состояния счёта. Если поддерживающая маржа равна 0, то используется начальная маржа.

Символ должен быть предварительно выбран методом [Name](#).

## MarginLong

Получает коэффициент взимания маржи по длинным позициям.

```
double MarginLong() const
```

### Возвращаемое значение

Коэффициент взимания маржи по длинным позициям.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## MarginShort

Получает коэффициент взимания маржи по коротким позициям.

```
double MarginShort() const
```

### Возвращаемое значение

Коэффициент взимания маржи по коротким позициям.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## MarginLimit

Получает коэффициент взимания маржи по Limit ордерам.

```
double MarginLimit() const
```

### Возвращаемое значение

Коэффициент взимания маржи по Limit ордерам.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## MarginStop

Получает коэффициент взимания маржи по Stop ордерам.

```
double MarginStop() const
```

### Возвращаемое значение

Коэффициент взимания маржи по Stop ордерам.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## MarginStopLimit

Получает коэффициент взимания маржи по Stop Limit ордерам.

```
double MarginStopLimit() const
```

### Возвращаемое значение

Коэффициент взимания маржи по Stop Limit ордерам.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TradeTimeFlags

Получает флаги разрешенных режимов истечения ордера.

```
int TradeTimeFlags() const
```

### Возвращаемое значение

Флаги разрешенных режимов истечения ордера.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TradeFillFlags

Получает флаги разрешенных режимов заливки ордера.

```
int TradeFillFlags() const
```

### Возвращаемое значение

Флаги разрешенных режимов заливки ордера.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Digits

Получает количество знаков после десятичной точки.

```
int Digits() const
```

### Возвращаемое значение

Количество знаков после десятичной точки.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Point

Получает значение одного пункта.

```
double Point() const
```

### Возвращаемое значение

Значение одного пункта.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TickCount

Получает стоимость минимального изменения цены.

```
double TickValue() const
```

### Возвращаемое значение

Стоимость минимального изменения цены.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TickValueProfit

Получает рассчитанную стоимость тика для прибыльной позиции.

```
double TickValueProfit() const
```

### Возвращаемое значение

Рассчитанная стоимость тика для прибыльной позиции.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TickValueLoss

Получает рассчитанную стоимость тика для убыточной позиции.

```
double TickValueLoss() const
```

### Возвращаемое значение

Рассчитанная стоимость тика для убыточной позиции.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## TickSize

Получает минимальное изменение цены.

```
double TickSize() const
```

### Возвращаемое значение

Минимальное изменение цены.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## ContractSize

Получает размер торгового контракта.

```
double ContractSize() const
```

### Возвращаемое значение

Размер торгового контракта.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## LotsMin

Получает минимальный объем для заключения сделки.

```
double LotsMin() const
```

### Возвращаемое значение

Минимальный объем для заключения сделки.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## LotsMax

Получает максимальный объем для заключения сделки.

```
double LotsMax() const
```

### Возвращаемое значение

Максимальный объем для заключения сделки.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## LotsStep

Получает минимальный шаг изменения объема для заключения сделки.

```
double LotsStep() const
```

### Возвращаемое значение

Минимальный шаг изменения объема для заключения сделки.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## LotsLimit

Получает максимально допустимый совокупный объем открытой позиции и отложенных ордеров (вне зависимости от направления) на одном символе.

```
double LotsLimit() const
```

### Возвращаемое значение

Максимально допустимый совокупный объем открытой позиции и отложенных ордеров (вне зависимости от направления) на одном символе.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SwapLong

Получает значение свопа длинной позиции.

```
double SwapLong() const
```

### Возвращаемое значение

Значение свопа длинной позиции.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SwapShort

Получает значение свопа короткой позиции.

```
double SwapShort() const
```

### Возвращаемое значение

Значение свопа короткой позиции.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## CurrencyBase

Получает наименование базовой валюты символа.

```
string CurrencyBase() const
```

### Возвращаемое значение

Наименование базовой валюты символа.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## CurrencyProfit

Получает наименование валюты прибыли.

```
string CurrencyProfit() const
```

### Возвращаемое значение

Наименование валюты прибыли.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## CurrencyMargin

Получает наименование валюты залога.

```
string CurrencyMargin() const
```

### Возвращаемое значение

Наименование валюты залога.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Bank

Получает наименование источника текущей котировки.

```
string Bank() const
```

### Возвращаемое значение

Наименование источника текущей котировки.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Description

Получает строковое описание символа.

```
string Description() const
```

### Возвращаемое значение

Строковое описание символа.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Path

Получает путь в дереве символов.

```
string Path() const
```

### Возвращаемое значение

Путь в дереве символов.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionDeals

Получает количество сделок в текущей сессии.

```
long SessionDeals() const
```

### Возвращаемое значение

Количество сделок в текущей сессии.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionBuyOrders

Получает общее число ордеров на покупку в текущий момент.

```
long SessionBuyOrders() const
```

### Возвращаемое значение

Общее число ордеров на покупку в текущий момент.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionSellOrders

Получает общее число ордеров на продажу в текущий момент.

```
long SessionSellOrders() const
```

### Возвращаемое значение

Общее число ордеров на продажу в текущий момент.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionTurnover

Получает суммарный оборот в текущую сессию.

```
double SessionTurnover() const
```

### Возвращаемое значение

Суммарный оборот в текущую сессию.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionInterest

Получает суммарный объем открытых позиций.

```
double SessionInterest() const
```

### Возвращаемое значение

Суммарный объем открытых позиций.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionBuyOrdersVolume

Получает общий объем ордеров на покупку в текущий момент.

```
double SessionBuyOrdersVolume() const
```

### Возвращаемое значение

Общий объем ордеров на покупку в текущий момент.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionSellOrdersVolume

Получает общий объем ордеров на продажу в текущий момент.

```
double SessionSellOrdersVolume() const
```

### Возвращаемое значение

Общий объем ордеров на продажу в текущий момент.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionOpen

Получает цену открытия текущей сессии.

```
double SessionOpen() const
```

### Возвращаемое значение

Цена открытия текущей сессии.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionClose

Получает цену закрытия текущей сессии.

```
double SessionClose() const
```

### Возвращаемое значение

Цена закрытия текущей сессии.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionAW

Получает значение средневзвешенной цены текущей сессии.

```
double SessionAW() const
```

### Возвращаемое значение

Значение средневзвешенной цены текущей сессии.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionPriceSettlement

Получает значение цены поставки на текущую сессию.

```
double SessionPriceSettlement() const
```

### Возвращаемое значение

Значение цены поставки на текущую сессию.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionPriceLimitMin

Получает минимально допустимое значение цены за текущую сессию.

```
double SessionPriceLimitMin() const
```

### Возвращаемое значение

Минимально допустимое значение цены за текущую сессию.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## SessionPriceLimitMax

Получает максимально допустимое значение цены на сессию.

```
double SessionPriceLimitMax() const
```

### Возвращаемое значение

Максимально допустимое значение цены на сессию.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## InfolInteger

Получает значение указанного целочисленного свойства.

```
bool InfoInteger(
    ENUM_SYMBOL_INFO_INTEGER prop_id,      // идентификатор свойства
    long&                      var        // ссылка на переменную
) const
```

### Параметры

*prop\_id*

[in] Идентификатор целочисленного свойства из перечисления [ENUM\\_SYMBOL\\_INFO\\_INTEGER](#).

*var*

[out] Ссылка на переменную типа [long](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## InfoDouble

Получает значение указанного double-свойства.

```
bool InfoDouble(
    ENUM_SYMBOL_INFO_DOUBLE prop_id,           // идентификатор свойства
    double&                  var              // ссылка на переменную
) const
```

### Параметры

*prop\_id*

[in] Идентификатор double-свойства из перечисления [ENUM\\_SYMBOL\\_INFO\\_DOUBLE](#).

*var*

[out] Ссылка на переменную типа [double](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## InfoString

Получает значение указанного текстового свойства.

```
bool InfoString(  
    ENUM_SYMBOL_INFO_STRING prop_id,      // идентификатор свойства  
    string&                  var        // ссылка на переменную  
) const
```

### Параметры

*prop\_id*

[in] Идентификатор текстового свойства.

*var*

[out] Ссылка на переменную типа [string](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## NormalizePrice

Нормализует цену с учетом свойств символа.

```
double NormalizePrice(  
    double price // цена  
) const
```

### Параметры

*price*

[in] Цена.

### Возвращаемое значение

Нормализованная цена.

### Примечание

Символ должен быть предварительно выбран методом [Name](#).

## Класс COrderInfo

Класс COrderInfo является классом для упрощенного доступа к свойствам отложенного ордера.

### Описание

Класс COrderInfo обеспечивает доступ к свойствам отложенного ордера.

### Декларация

```
class COrderInfo : public CObject
```

### Заголовок

```
#include <Trade\OrderInfo.mqh>
```

### Иерархия наследования

[CObject](#)

COrderInfo

### Методы класса по группам

Доступ к целочисленным свойствам	
<a href="#"><u>Ticket</u></a>	Получает тикет ордера, предварительно выбранного для доступа
<a href="#"><u>TimeSetup</u></a>	Получает время установки ордера
<a href="#"><u>TimeSetupMsc</u></a>	Получает время установки ордера в миллисекундах с 01.01.1970
<a href="#"><u>OrderType</u></a>	Получает тип ордера
<a href="#"><u>TypeDescription</u></a>	Получает тип ордера как строку
<a href="#"><u>State</u></a>	Получает статус ордера
<a href="#"><u>StateDescription</u></a>	Получает статус ордера как строку
<a href="#"><u>TimeExpiration</u></a>	Получает время истечения ордера
<a href="#"><u>TimeDone</u></a>	Получает время исполнения или снятия ордера
<a href="#"><u>TimeDoneMsc</u></a>	Получает время исполнения или снятия ордера в миллисекундах с 01.01.1970
<a href="#"><u>TypeFilling</u></a>	Получает тип исполнения ордера по остатку
<a href="#"><u>TypeFillingDescription</u></a>	Получает тип исполнения ордера по остатку как строку
<a href="#"><u>TypeTime</u></a>	Получает тип ордера по времени истечения

<a href="#">TypeTimeDescription</a>	Получает тип ордера по времени истечения как строку
<a href="#">Magic</a>	Получает идентификатор эксперта, выставившего ордер
<a href="#">PositionId</a>	Получает идентификатор позиции
<b>Доступ к double-свойствам</b>	
<a href="#">VolumeInitial</a>	Получает первоначальный объем ордера
<a href="#">VolumeCurrent</a>	Получает невыполненный объем ордера
<a href="#">PriceOpen</a>	Получает цену ордера
<a href="#">StopLoss</a>	Получает стоп лосс ордера
<a href="#">TakeProfit</a>	Получает тейк профит ордера
<a href="#">PriceCurrent</a>	Получает текущую цену по символу ордера
<a href="#">PriceStopLimit</a>	Получает цену постановки лимит ордера
<b>Доступ к текстовым свойствам</b>	
<a href="#">Symbol</a>	Получает наименование символа ордера
<a href="#">Comment</a>	Получает комментарий ордера
<b>Доступ к функциям API MQL5</b>	
<a href="#">InfoInteger</a>	Получает значение указанного целочисленного свойства
<a href="#">InfoDouble</a>	Получает значение указанного double-свойства
<a href="#">InfoString</a>	Получает значение указанного текстового свойства
<b>Состояние</b>	
<a href="#">StoreState</a>	Сохраняет параметры ордера
<a href="#">CheckState</a>	Сравнивает текущие параметры с сохраненными
<b>Выбор</b>	
<a href="#">Select</a>	Выбирает ордер для дальнейшего доступа по указанному тикету
<a href="#">SelectByIndex</a>	Выбирает ордер для дальнейшего доступа по указанному индексу

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## Ticket

Получает тикет ордера.

```
ulong Ticket() const
```

### Возвращаемое значение

Тикет ордера в случае успеха, или [ULONG\\_MAX](#) в случае ошибки.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TimeSetup

Получает время установки ордера.

```
datetime TimeSetup() const
```

### Возвращаемое значение

Время установки ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TimeSetupMsc

Получает время установки ордера на исполнение в миллисекундах с 01.01.1970.

```
ulong TimeSetupMsc() const
```

### Возвращаемое значение

Время установки ордера на исполнение в миллисекундах с 01.01.1970.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## OrderType

Получает тип ордера.

```
ENUM_ORDER_TYPE OrderType()
```

### Возвращаемое значение

Тип ордера из перечисления [ENUM\\_ORDER\\_TYPE](#).

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeDescription

Получает тип ордера как строку.

```
string TypeDescription() const
```

### Возвращаемое значение

Тип ордера как строка.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## State

Получает статус ордера.

```
ENUM_ORDER_STATE State() const
```

### Возвращаемое значение

Статус ордера из перечисления [ENUM\\_ORDER\\_STATE](#).

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## StateDescription

Получает статус ордера как строку.

```
string StateDescription() const
```

### Возвращаемое значение

Статус ордера как строка.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TimeExpiration

Получает время истечения ордера.

```
datetime TimeExpiration() const
```

### Возвращаемое значение

Заданное при установке время истечения ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TimeDone

Получает время исполнения или снятия ордера.

```
datetime TimeDone() const
```

### Возвращаемое значение

Время исполнения или снятия ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TimeDoneMsc

Получает время исполнения или снятия ордера в миллисекундах с 01.01.1970.

```
ulong TimeDoneMsc() const
```

### Возвращаемое значение

Время исполнения или снятия ордера в миллисекундах с 01.01.1970.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeFilling

Получает тип исполнения ордера по остатку.

```
ENUM_ORDER_TYPE_FILLING TypeFilling() const
```

### Возвращаемое значение

Тип исполнения ордера по остатку из перечисления [ENUM\\_ORDER\\_TYPE\\_FILLING](#).

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeFillingDescription

Получает тип исполнения ордера по остатку как строку.

```
string TypeFillingDescription() const
```

### Возвращаемое значение

Тип исполнения ордера по остатку как строка.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeTime

Получает тип ордера по времени истечения.

```
ENUM_ORDER_TYPE_TIME TypeTime() const
```

### Возвращаемое значение

Тип ордера по времени истечения из перечисления [ENUM\\_ORDER\\_TYPE\\_TIME](#).

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeTimeDescription

Получает тип ордера по времени истечения как строку.

```
string TypeTimeDescription() const
```

### Возвращаемое значение

Тип ордера по времени истечения как строка.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Magic

Получает идентификатор эксперта, выставившего ордер.

```
long Magic() const
```

### Возвращаемое значение

Идентификатор эксперта, выставившего ордер.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## PositionId

Получает идентификатор позиции.

```
long PositionId() const
```

### Возвращаемое значение

Идентификатор позиции, в которой участвует ордер.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## VolumInitial

Получает первоначальный объем ордера.

```
double VolumeInitial() const
```

### Возвращаемое значение

Первоначальный объем ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## VolumeCurrent

Получает невыполненный объем ордера.

```
double VolumeCurrent() const
```

### Возвращаемое значение

Невыполненный объем ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## PriceOpen

Получает цену ордера.

```
double PriceOpen() const
```

### Возвращаемое значение

Цена установки ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## StopLoss

Получает цену Stop Loss ордера.

```
double StopLoss() const
```

### Возвращаемое значение

Цена Stop Loss ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TakeProfit

Получает цену Take Profit ордера.

```
double TakeProfit() const
```

### Возвращаемое значение

Цена Take Profit ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## PriceCurrent

Получает текущую цену по символу ордера.

```
double PriceCurrent() const
```

### Возвращаемое значение

Текущая цена по символу ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## PriceStopLimit

Получает цену постановки отложенного ордера.

```
double PriceStopLimit() const
```

### Возвращаемое значение

Цена постановки отложенного ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Symbol

Получает наименование символа ордера.

```
string Symbol() const
```

### Возвращаемое значение

Наименование символа ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Comment

Получает комментарий ордера.

```
string Comment() const
```

### Возвращаемое значение

Комментарий ордера.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## InfolInteger

Получает значение указанного целочисленного свойства.

```
bool InfoInteger(
    ENUM_ORDER_PROPERTY_INTEGER prop_id,           // идентификатор свойства
    long& var                                     // ссылка на переменную
) const
```

### Параметры

*prop\_id*  
[in] Идентификатор целочисленного свойства из перечисления [ENUM\\_ORDER\\_PROPERTY\\_INTEGER](#).

*var*  
[out] Ссылка на переменную типа [long](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## InfoDouble

Получает значение указанного double-свойства.

```
bool InfoDouble(
    ENUM_ORDER_PROPERTY_DOUBLE prop_id,      // идентификатор свойства
    double&                      var        // ссылка на переменную
) const
```

### Параметры

*prop\_id*

[in] Идентификатор double-свойства из перечисления [ENUM\\_ORDER\\_PROPERTY\\_DOUBLE](#).

*var*

[out] Ссылка на переменную типа [double](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## InfoString

Получает значение указанного текстового свойства.

```
bool InfoString(  
    ENUM_ORDER_PROPERTY_STRING prop_id, // идентификатор свойства  
    string& var // ссылка на переменную  
) const
```

### Параметры

*prop\_id*

[in] Идентификатор текстового свойства из перечисления [ENUM\\_ORDER\\_PROPERTY\\_STRING](#).

*var*

[out] Ссылка на переменную типа [string](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Ордер должен быть предварительно выбран для доступа методом [Select](#) (по тикету) или [SelectByIndex](#) (по индексу).

## StoreState

Сохраняет параметры ордера.

```
void StoreState()
```

**Возвращаемое значение**

Нет.

## CheckState

Сравнивает текущие параметры с сохраненными.

```
bool CheckState()
```

### Возвращаемое значение

true - если параметры ордера изменились за время, прошедшее после последнего вызова метода [StoreState\(\)](#), иначе false.

## Select

Выбирает ордер для дальнейшего доступа по указанному тикету.

```
bool Select(  
    ulong ticket // тикет ордера  
)
```

### Возвращаемое значение

Возвращает true в случае успеха, или false, если ордер не выбран.

## SelectByIndex

Выбирает ордер по индексу для дальнейшей работы.

```
bool SelectByIndex(  
    int index // индекс ордера  
)
```

### Параметры

*index*

[in] Индекс ордера.

### Возвращаемое значение

Возвращает true в случае успеха, или false, если ордер не выбран.

## Класс CHistoryOrderInfo

Класс CHistoryOrderInfo является классом для упрощенного доступа к свойствам ордера истории.

### Описание

Класс CHistoryOrderInfo обеспечивает доступ к свойствам ордера истории.

### Декларация

```
class CHistoryOrderInfo : public CObject
```

### Заголовок

```
#include <Trade\HistoryOrderInfo.mqh>
```

### Иерархия наследования

CObject

CHistoryOrderInfo

### Методы класса по группам

Доступ к целочисленным свойствам	
<a href="#"><u>TimeSetup</u></a>	Получает время установки ордера
<a href="#"><u>TimeSetupMsc</u></a>	Получает время установки ордера в миллисекундах с 01.01.1970
<a href="#"><u>OrderType</u></a>	Получает тип ордера
<a href="#"><u>TypeDescription</u></a>	Получает тип ордера как строку
<a href="#"><u>State</u></a>	Получает статус ордера
<a href="#"><u>StateDescription</u></a>	Получает статус ордера как строку
<a href="#"><u>TimeExpiration</u></a>	Получает время истечения ордера
<a href="#"><u>TimeDone</u></a>	Получает время исполнения или снятия ордера
<a href="#"><u>TimeDoneMsc</u></a>	Получает время исполнения или снятия ордера в миллисекундах с 01.01.1970
<a href="#"><u>TypeFilling</u></a>	Получает тип исполнения ордера по остатку
<a href="#"><u>TypeFillingDescription</u></a>	Получает тип исполнения ордера по остатку как строку
<a href="#"><u>TypeTime</u></a>	Получает тип ордера по времени истечения
<a href="#"><u>TypeTimeDescription</u></a>	Получает тип ордера по времени истечения как строку

<a href="#">Magic</a>	Получает идентификатор эксперта, выставившего ордер
<a href="#">PositionId</a>	Получает идентификатор позиции
<b>Доступ к double-свойствам</b>	
<a href="#">VolumeInitial</a>	Получает первоначальный объем ордера
<a href="#">VolumeCurrent</a>	Получает невыполненный объем ордера
<a href="#">PriceOpen</a>	Получает цену ордера
<a href="#">StopLoss</a>	Получает стоп лосс ордера
<a href="#">TakeProfit</a>	Получает тейк профит ордера
<a href="#">PriceCurrent</a>	Получает текущую цену по символу ордера
<a href="#">PriceStopLimit</a>	Получает цену постановки лимит ордера
<b>Доступ к текстовым свойствам</b>	
<a href="#">Symbol</a>	Получает символ ордера
<a href="#">Comment</a>	Получает комментарий ордера
<b>Доступ к функциям API MQL5</b>	
<a href="#">InfoInteger</a>	Получает значение указанного целочисленного свойства
<a href="#">InfoDouble</a>	Получает значение указанного double-свойства
<a href="#">InfoString</a>	Получает значение указанного текстового свойства
<b>Выбор</b>	
<a href="#">Ticket</a>	Получает тикет/выбирает ордер
<a href="#">SelectByIndex</a>	Выбирает исторический ордер по индексу

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## TimeSetup

Получает время установки ордера.

```
datetime TimeSetup() const
```

### Возвращаемое значение

Время установки ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TimeSetupMsc

Получает время установки ордера на исполнение в миллисекундах с 01.01.1970.

```
ulong TimeSetupMsc() const
```

### Возвращаемое значение

Время установки ордера на исполнение в миллисекундах с 01.01.1970.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## OrderType

Получает тип ордера.

```
ENUM_ORDER_TYPE OrderType() const
```

### Возвращаемое значение

Тип ордера из перечисления [ENUM\\_ORDER\\_TYPE](#).

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeDescription

Получает тип ордера как строку.

```
string TypeDescription() const
```

### Возвращаемое значение

Тип ордера как строка.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## State

Получает статус ордера.

```
ENUM_ORDER_STATE State() const
```

### Возвращаемое значение

Статус ордера из перечисления [ENUM\\_ORDER\\_STATE](#).

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## StateDescription

Получает статус ордера как строку.

```
string StateDescription() const
```

### Возвращаемое значение

Статус ордера как строка.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TimeExpiration

Получает время истечения ордера.

```
datetime TimeExpiration() const
```

### Возвращаемое значение

Заданное при установке время истечения ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TimeDone

Получает время исполнения или снятия ордера.

```
datetime TimeDone() const
```

### Возвращаемое значение

Время исполнения или снятия ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TimeDoneMsc

Получает время исполнения или снятия ордера в миллисекундах с 01.01.1970.

```
ulong TimeDoneMsc() const
```

### Возвращаемое значение

Время исполнения или снятия ордера в миллисекундах с 01.01.1970.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeFilling

Получает тип исполнения ордера по остатку.

```
ENUM_ORDER_TYPE_FILLING TypeFilling() const
```

### Возвращаемое значение

Тип исполнения ордера по остатку из перечисления [ENUM\\_ORDER\\_TYPE\\_FILLING](#).

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeFillingDescription

Получает тип исполнения ордера по остатку как строку.

```
string TypeFillingDescription() const
```

### Возвращаемое значение

Тип исполнения ордера по остатку как строка.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeTime

Получает тип ордера по времени истечения.

```
ENUM_ORDER_TYPE_TIME TypeTime() const
```

### Возвращаемое значение

Тип ордера по времени истечения из перечисления [ENUM\\_ORDER\\_TYPE\\_TIME](#).

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeTimeDescription

Получает тип ордера по времени истечения как строку.

```
string TypeTimeDescription() const
```

### Возвращаемое значение

Тип ордера по времени истечения как строка.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Magic

Получает идентификатор эксперта, выставившего ордер.

```
long Magic() const
```

### Возвращаемое значение

Идентификатор эксперта, выставившего ордер.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## PositionId

Получает идентификатор позиции.

```
long PositionId() const
```

### Возвращаемое значение

Идентификатор позиции, в которой участвовал ордер.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## VolumInitial

Получает первоначальный объем ордера.

```
double VolumeInitial() const
```

### Возвращаемое значение

Первоначальный объем ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## VolumeCurrent

Получает невыполненный объем ордера.

```
double VolumeCurrent() const
```

### Возвращаемое значение

Невыполненный объем ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## PriceOpen

Получает цену ордера.

```
double PriceOpen() const
```

### Возвращаемое значение

Цена установки ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## StopLoss

Получает цену Stop Loss ордера.

```
double StopLoss() const
```

### Возвращаемое значение

Цена Stop Loss ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TakeProfit

Получает цену Take Profit ордера.

```
double TakeProfit() const
```

### Возвращаемое значение

Цена Take Profit ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## PriceCurrent

Получает текущую цену по символу ордера.

```
double PriceCurrent() const
```

### Возвращаемое значение

Текущая цена по символу ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## PriceStopLimit

Получает цену постановки отложенного ордера.

```
double PriceStopLimit() const
```

### Возвращаемое значение

Цена постановки отложенного ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Symbol

Получает наименование символа ордера.

```
string Symbol() const
```

### Возвращаемое значение

Наименование символа ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Comment

Получает комментарий ордера.

```
string Comment() const
```

### Возвращаемое значение

Комментарий ордера.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## InfolInteger

Получает значение указанного целочисленного свойства.

```
bool InfoInteger(
    ENUM_ORDER_PROPERTY_INTEGER prop_id,           // идентификатор свойства
    long& var                                     // ссылка на переменную
) const
```

### Параметры

*prop\_id*  
[in] Идентификатор целочисленного свойства из перечисления [ENUM\\_ORDER\\_PROPERTY\\_INTEGER](#).

*var*  
[out] Ссылка на переменную типа [long](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## InfoDouble

Получает значение указанного double-свойства.

```
bool InfoDouble(
    ENUM_ORDER_PROPERTY_DOUBLE prop_id,      // идентификатор свойства
    double&                      var        // ссылка на переменную
) const
```

### Параметры

*prop\_id*

[in] Идентификатор double-свойства из перечисления [ENUM\\_ORDER\\_PROPERTY\\_DOUBLE](#).

*var*

[out] Ссылка на переменную типа [double](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## InfoString

Получает значение указанного текстового свойства.

```
bool InfoString(
    ENUM_ORDER_PROPERTY_STRING prop_id,      // идентификатор свойства
    string&                      var        // ссылка на переменную
) const
```

### Параметры

*prop\_id*

[in] Идентификатор текстового свойства из перечисления [ENUM\\_ORDER\\_PROPERTY\\_STRING](#).

*var*

[out] Ссылка на переменную типа [string](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Исторический ордер должен быть предварительно выбран для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Ticket (метод Get)

Получает тикет ордера.

```
ulong Ticket() const
```

### Возвращаемое значение

Тикет ордера.

## Ticket (метод Set)

Выбирает ордер для дальнейшей работы.

```
void Ticket(  
    ulong ticket // тикет  
)
```

### Параметры

*ticket*

[in] Тикет ордера.

## SelectByIndex

Выбирает исторический ордер по индексу для дальнейшей работы.

```
bool SelectByIndex(  
    int index // индекс ордера  
)
```

### Параметры

*index*

[in] Индекс исторического ордера.

### Возвращаемое значение

Возвращает true в случае успеха, или false, если ордер не выбран.

## Класс CPositionInfo

Класс CPositionInfo является классом для упрощенного доступа к свойствам открытой рыночной позиции.

### Описание

Класс CPositionInfo обеспечивает доступ к свойствам открытой рыночной позиции.

### Декларация

```
class CPositionInfo : public CObject
```

### Заголовок

```
#include <Trade\PositionInfo.mqh>
```

### Иерархия наследования

[CObject](#)

CPositionInfo

### Методы класса по группам

Доступ к целочисленным свойствам	
<a href="#"><u>Time</u></a>	Получает время открытия позиции
<a href="#"><u>TimeMsc</u></a>	Получает время открытия позиции в миллисекундах с 01.01.1970
<a href="#"><u>TimeUpdate</u></a>	Получает время изменения позиции в секундах с 01.01.1970
<a href="#"><u>TimeUpdateMsc</u></a>	Получает время изменения позиции в миллисекундах с 01.01.1970
<a href="#"><u>PositionType</u></a>	Получает тип позиции
<a href="#"><u>TypeDescription</u></a>	Получает тип позиции как строку
<a href="#"><u>Magic</u></a>	Получает идентификатор эксперта, открывшего позицию
<a href="#"><u>Identifier</u></a>	Получает идентификатор позиции
Доступ к double-свойствам	
<a href="#"><u>Volume</u></a>	Получает объем позиции
<a href="#"><u>PriceOpen</u></a>	Получает цену открытия позиции
<a href="#"><u>StopLoss</u></a>	Получает цену стоп лосса позиции
<a href="#"><u>TakeProfit</u></a>	Получает цену тейк профита позиции

<a href="#">PriceCurrent</a>	Получает текущую цену по символу позиции
<a href="#">Commission</a>	Получает размер комиссии по позиции
<a href="#">Swap</a>	Получает размер свопа по позиции
<a href="#">Profit</a>	Получает размер текущей прибыли по позиции
<b>Доступ к текстовым свойствам</b>	
<a href="#">Symbol</a>	Получает наименование символа позиции
<a href="#">Comment</a>	Получает комментарий позиции
<b>Доступ к функциям API MQL5</b>	
<a href="#">InfoInteger</a>	Получает значение указанного целочисленного свойства
<a href="#">InfoDouble</a>	Получает значение указанного double-свойства
<a href="#">InfoString</a>	Получить значение указанного текстового свойства
<b>Выбор</b>	
<a href="#">Select</a>	Выбирает позицию
<a href="#">SelectByIndex</a>	Выбирает позицию по индексу
<a href="#">SelectByMagic</a>	Выбирает позицию по имени инструмента и магическому номеру для дальнейшей работы
<a href="#">SelectByTicket</a>	Выбирает позицию по тикету
<b>Состояние</b>	
<a href="#">StoreState</a>	Сохраняет параметры позиции
<a href="#">CheckState</a>	Сравнивает текущие параметры с сохраненными

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## Time

Получает время открытия позиции.

```
datetime Time() const
```

### Возвращаемое значение

Время открытия позиции.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## TimeMsc

Получает время открытия позиции в миллисекундах с 01.01.1970.

```
ulong TimeMsc() const
```

### Возвращаемое значение

Время открытия позиции в миллисекундах с 01.01.1970.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## TimeUpdate

Получает время изменения позиции в секундах с 01.01.1970.

```
datetime TimeUpdate() const
```

### Возвращаемое значение

Время изменения позиции в секундах с 01.01.1970.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## TimeUpdateMsc

Получает время изменения позиции в миллисекундах с 01.01.1970.

```
ulong TimeUpdateMsc() const
```

### Возвращаемое значение

Время изменения позиции в миллисекундах с 01.01.1970.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## PositionType

Получает тип позиции.

```
ENUM_POSITION_TYPE PositionType() const
```

### Возвращаемое значение

Тип позиции из перечисления [ENUM\\_POSITION\\_TYPE](#).

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## TypeDescription

Получает тип позиции как строку.

```
string TypeDescription() const
```

### Возвращаемое значение

Тип позиции как строка.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## Magic

Получает идентификатор эксперта, открывшего позицию.

```
long Magic() const
```

### Возвращаемое значение

Идентификатор эксперта, открывшего позицию.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## Identifier

Получает идентификатор позиции.

```
long Identifier() const
```

### Возвращаемое значение

Идентификатор позиции.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## Volume

Получает объем позиции.

```
double Volume() const
```

### Возвращаемое значение

Объем позиции.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## PriceOpen

Получает цену открытия позиции.

```
double PriceOpen() const
```

### Возвращаемое значение

Цена открытия позиции.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## StopLoss

Получает цену Stop Loss позиции.

```
double StopLoss() const
```

### Возвращаемое значение

Цена Stop Loss позиции.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## TakeProfit

Получает цену Take Profit позиции.

```
double TakeProfit() const
```

### Возвращаемое значение

Цена Take Profit позиции.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## PriceCurrent

Получает текущую цену по символу позиции.

```
double PriceCurrent() const
```

### Возвращаемое значение

Текущая цена по символу позиции.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## Commission

Получает размер комиссии по позиции.

```
double Commission() const
```

### Возвращаемое значение

Размер комиссии по позиции в валюте депозита.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## Swap

Получает размер свопа по позиции.

```
double Swap() const
```

### Возвращаемое значение

Размер свопа по позиции в валюте депозита.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## Profit

Получает размер текущей прибыли по позиции.

```
double Profit() const
```

### Возвращаемое значение

Размер текущей прибыли по позиции в валюте депозита.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## Symbol

Получает наименование символа позиции.

```
string Symbol() const
```

### Возвращаемое значение

Наименование символа позиции.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## Comment

Получает комментарий позиции.

```
string Comment() const
```

### Возвращаемое значение

Комментарий позиции.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## InfolInteger

Получает значение указанного целочисленного свойства.

```
bool InfoInteger(
    ENUM_POSITION_PROPERTY_INTEGER prop_id,      // идентификатор свойства
    long& var                                     // ссылка на переменную
) const
```

### Параметры

*prop\_id*  
[in] Идентификатор целочисленного свойства из перечисления [ENUM\\_POSITION\\_PROPERTY\\_INTEGER](#).

*var*  
[out] Ссылка на переменную типа [long](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## InfoDouble

Получает значение указанного double-свойства.

```
bool InfoDouble(
    ENUM_POSITION_PROPERTY_DOUBLE prop_id,           // идентификатор свойства
    double&                         var            // ссылка на переменную
) const
```

### Параметры

*prop\_id*

[in] Идентификатор double-свойства из перечисления [ENUM\\_POSITION\\_PROPERTY\\_DOUBLE](#).

*var*

[out] Ссылка на переменную типа [double](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## InfoString

Получает значение указанного текстового свойства.

```
bool InfoString(  
    ENUM_POSITION_PROPERTY_STRING prop_id,      // идентификатор свойства  
    string&                         var        // ссылка на переменную  
) const
```

### Параметры

*prop\_id*

[in] Идентификатор текстового свойства из перечисления [ENUM\\_POSITION\\_PROPERTY\\_STRING](#).

*var*

[out] Ссылка на переменную типа [string](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Позиция должна быть предварительно выбрана для доступа методом [Select](#) (по символу) или [SelectByIndex](#) (по индексу).

## Select

Выбирает позицию для дальнейшей работы.

```
bool Select(  
    const string symbol // символ  
)
```

### Параметры

*symbol*

[in] Символ для выбора позиции.

## SelectByIndex

Выбирает позицию по индексу для дальнейшей работы.

```
bool SelectByIndex(  
    int index // индекс позиции  
)
```

### Параметры

*index*

[in] Индекс позиции.

### Возвращаемое значение

Возвращает true в случае успеха, или false, если позиция не выбрана.

## SelectByMagic

Выбирает позицию по имени инструмента и магическому номеру для дальнейшей работы.

```
bool SelectByMagic(  
    const string symbol, // имя инструмента  
    const ulong magic   // магический номер  
);
```

### Параметры

*symbol*

[in] Имя инструмента.

*magic*

[in] Магический номер позиции.

### Возвращаемое значение

Возвращает true в случае успеха, или false, если позиция не выбрана.

## SelectByTicket

Выбирает позицию по тикету для дальнейшей работы.

```
bool SelectByTicket(  
    ulong ticket           // тикет позиции  
)
```

### Параметры

*ticket*

[in] Тикет позиции.

### Возвращаемое значение

Возвращает true в случае успеха, или false, если позиция не выбрана.

## StoreState

Сохраняет параметры позиции.

```
void StoreState()
```

### Возвращаемое значение

Нет.

## CheckState

Сравнивает текущие параметры с сохраненными.

```
bool CheckState()
```

### Возвращаемое значение

true - если параметры позиции изменились за время, прошедшее после последнего вызова метода [StoreState\(\)](#), иначе false.

## Класс CDealInfo

Класс CDealInfo является классом для упрощенного доступа к свойствам сделки.

### Описание

Класс CDealInfo обеспечивает доступ к свойствам сделки.

### Декларация

```
class CDealInfo : public CObject
```

### Заголовок

```
#include <Trade\DealInfo.mqh>
```

### Иерархия наследования

[CObject](#)

CDealInfo

### Методы класса по группам

Доступ к целочисленным свойствам	
<a href="#"><u>Order</u></a>	Получает ордер, на основании которого выполнена сделка
<a href="#"><u>Time</u></a>	Получает время совершения сделки
<a href="#"><u>TimeMsc</u></a>	Получает время совершения сделки в миллисекундах с 01.01.1970
<a href="#"><u>DealType</u></a>	Получает тип сделки
<a href="#"><u>TypeDescription</u></a>	Получает тип сделки как строку
<a href="#"><u>Entry</u></a>	Получает направление сделки
<a href="#"><u>EntryDescription</u></a>	Получает направление сделки как строку
<a href="#"><u>Magic</u></a>	Получает идентификатор эксперта, совершившего сделку
<a href="#"><u>PositionId</u></a>	Получает идентификатор позиции, в которой участвовала сделка
Доступ к double-свойствам	
<a href="#"><u>Volume</u></a>	Получает объем сделки
<a href="#"><u>Price</u></a>	Получает цену сделки
<a href="#"><u>Commision</u></a>	Получает размер комиссии по сделке
<a href="#"><u>Swap</u></a>	Получает размер свопа при закрытии позиции

<a href="#"><u>Profit</u></a>	Получает финансовый результат сделки
<b>Доступ к текстовым свойствам</b>	
<a href="#"><u>Symbol</u></a>	Получает наименование символа сделки
<a href="#"><u>Comment</u></a>	Получает комментарий к сделке
<b>Доступ к функциям API MQL5</b>	
<a href="#"><u>InfoInteger</u></a>	Получает значение указанного целочисленного свойства
<a href="#"><u>InfoDouble</u></a>	Получает значение указанного double-свойства
<a href="#"><u>InfoString</u></a>	Получает значение указанного текстового свойства
<b>Выбор</b>	
<a href="#"><u>Ticket</u></a>	Получает тикет/выбирает сделку
<a href="#"><u>SelectByIndex</u></a>	Выбирает сделку по индексу

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## Order

Получает ордер, на основании которого выполнена сделка.

```
long Order() const
```

### Возвращаемое значение

Ордер, на основании которого выполнена сделка.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Time

Получает время совершения сделки.

```
datetime Time() const
```

### Возвращаемое значение

Время совершения сделки.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TimeMsc

Получает время совершения сделки в миллисекундах с 01.01.1970.

```
ulong TimeMsc() const
```

### Возвращаемое значение

Время совершения сделки в миллисекундах с 01.01.1970.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## DealType

Получает тип сделки.

```
ENUM DEAL TYPE DealType() const
```

### Возвращаемое значение

Тип сделки из перечисления [ENUM DEAL TYPE](#).

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## TypeDescription

Получает тип сделки как строку.

```
string TypeDescription() const
```

### Возвращаемое значение

Тип сделки как строка.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Entry

Сравнивает направление сделки.

```
ENUM_DEAL_ENTRY Entry() const
```

### Возвращаемое значение

Направление сделки (значение перечисления [ENUM DEAL ENTRY](#)).

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## EntryDescription

Получает направление сделки как строку.

```
string EntryDescription() const
```

### Возвращаемое значение

Направление сделки как строка.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Magic

Получает идентификатор эксперта, совершившего сделку.

```
long Magic() const
```

### Возвращаемое значение

Идентификатор эксперта, совершившего сделку.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## PositionId

Получает идентификатор позиции, в которой участвовала сделка.

```
long PositionId() const
```

### Возвращаемое значение

Идентификатор позиции, в которой участвовала сделка.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Volume

Получает объем сделки.

```
double Volume() const
```

### Возвращаемое значение

Объем сделки.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Price

Получает цену сделки.

```
double Price() const
```

### Возвращаемое значение

Цена сделки.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Commission

Получает размер комиссии по сделке.

```
double Commission() const
```

### Возвращаемое значение

Размер комиссии по сделке.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Swap

Получает размер свопа при закрытии позиции.

```
double Swap() const
```

### Возвращаемое значение

Размер свопа при закрытии позиции.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Profit

Получает финансовый результат сделки.

```
double Profit() const
```

### Возвращаемое значение

Финансовый результат сделки в валюте депозита.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Symbol

Получает наименование символа сделки.

```
string Symbol() const
```

### Возвращаемое значение

Наименование символа сделки.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Comment

Получает комментарий к сделке.

```
string Comment() const
```

### Возвращаемое значение

Комментарий к сделке.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## InfolInteger

Получает значение указанного целочисленного свойства.

```
bool InfoInteger(
    ENUM DEAL PROPERTY INTEGER prop_id,      // идентификатор свойства
    long& var                                // ссылка на переменную
) const
```

### Параметры

*prop\_id*  
[in] Идентификатор целочисленного свойства из перечисления [ENUM DEAL PROPERTY INTEGER](#).

*var*  
[out] Ссылка на переменную типа [long](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## InfoDouble

Получает значение указанного double-свойства.

```
bool InfoDouble(
    ENUM_DEAL_PROPERTY_DOUBLE prop_id,           // идентификатор свойства
    double&                      var            // ссылка на переменную
) const
```

### Параметры

*prop\_id*

[in] Идентификатор double-свойства из перечисления [ENUM DEAL PROPERTY DOUBLE](#).

*var*

[in] Ссылка на переменную типа [double](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## InfoString

Получает значение указанного текстового свойства.

```
bool InfoString(  
    ENUM DEAL PROPERTY STRING prop_id,           // идентификатор свойства  
    string& var                                // ссылка на переменную  
) const
```

### Параметры

*prop\_id*

[in] Идентификатор текстового свойства из перечисления [ENUM DEAL PROPERTY STRING](#).

*var*

[out] Ссылка на переменную типа [string](#) для размещения результата.

### Возвращаемое значение

true - в случае удачи, false - если не удалось получить значение свойства.

### Примечание

Сделка должна быть предварительно выбрана для доступа методом [Ticket](#) (по тикету) или [SelectByIndex](#) (по индексу).

## Ticket (метод Get)

Получает тикет сделки.

```
ulong Ticket() const
```

### Возвращаемое значение

Тикет сделки.

## Ticket (метод Set)

Выбирает сделку для дальнейшей работы.

```
void Ticket(  
    ulong ticket // тикет  
)
```

### Параметры

*ticket*

[in] Тикет сделки.

## SelectByIndex

Выбирает сделку по индексу для дальнейшей работы.

```
bool SelectByIndex(
    int index           // индекс сделки
)
```

### Параметры

*index*

[in] Индекс сделки.

### Возвращаемое значение

Возвращает true в случае успеха, или false, если сделка не выбрана.

## Класс CTrade

Класс CTrade является классом для упрощенного доступа к торговым функциям.

### Описание

Класс CTrade обеспечивает упрощенный доступ к торговым функциям.

### Декларация

```
class CTrade : public CObject
```

### Заголовок

```
#include <Trade\Trade.mqh>
```

### Иерархия наследования

[CObject](#)

CTrade

### Прямые потомки

[CExpertTrade](#)

### Методы класса по группам

Установка параметров	
<a href="#">LogLevel</a>	Устанавливает уровень логирования сообщений
<a href="#">SetExpertMagicNumber</a>	Устанавливает идентификатор эксперта
<a href="#">SetDeviationInPoints</a>	Устанавливает допустимое проскальзывание
<a href="#">SetTypeFilling</a>	Устанавливает тип ордера по исполнению
<a href="#">SetTypeFillingBySymbol</a>	Устанавливает тип ордера по исполнению согласно настройкам указанного символа
<a href="#">SetAsyncMode</a>	Устанавливает асинхронный режим торговых операций
<a href="#">SetMarginMode</a>	Устанавливает режим расчета маржи в соответствии с настройками текущего счета
Операции с ордерами	
<a href="#">OrderOpen</a>	Размещает отложенный ордер с заданными параметрами
<a href="#">OrderModify</a>	Изменяет параметры отложенного ордера
<a href="#">OrderDelete</a>	Удаляет отложенный ордер

<b>Операции с позициями</b>	
<a href="#">PositionOpen</a>	Открывает позицию с заданными параметрами
<a href="#">PositionModify</a>	Изменяет параметры позиции по указанному символу или тикету позиции
<a href="#">PositionClose</a>	Закрывает позицию по указанному символу
<a href="#">PositionClosePartial</a>	Закрывает часть позиции по указанному символу или с указанным тикетом
<a href="#">PositionCloseBy</a>	Закрывает позицию с указанным тикетом позицией встречного направления
<b>Дополнительные методы</b>	
<a href="#">Buy</a>	Открывает длинную позицию с заданными параметрами
<a href="#">Sell</a>	Открывает короткую позицию с заданными параметрами
<a href="#">BuyLimit</a>	Устанавливает отложенный ордер на покупку по цене лучше рыночной
<a href="#">BuyStop</a>	Устанавливает отложенный ордер на покупку по цене хуже рыночной
<a href="#">SellLimit</a>	Устанавливает отложенный ордер на продажу по цене лучше рыночной
<a href="#">SellStop</a>	Устанавливает отложенный ордер на продажу по цене хуже рыночной
<b>Доступ к параметрам последнего запроса</b>	
<a href="#">Request</a>	Получает копию структуры последнего запроса
<a href="#">RequestAction</a>	Получает тип торговой операции
<a href="#">RequestActionDescription</a>	Получает тип торговой операции как строку
<a href="#">RequestMagic</a>	Получает идентификатор эксперта
<a href="#">RequestOrder</a>	Получает тикет ордера
<a href="#">RequestSymbol</a>	Получает имя торгового инструмента, по которому выставляется ордер
<a href="#">RequestVolume</a>	Получает запрашиваемый объем сделки в лотах
<a href="#">RequestPrice</a>	Получает цену, при достижении которой ордер должен быть исполнен

<a href="#">RequestStopLimit</a>	Получает цену, по которой будет выставлен отложенный Stop Limit ордер
<a href="#">RequestSL</a>	Получает цену, по которой сработает Stop Loss
<a href="#">RequestTP</a>	Получает цену, по которой сработает Take Profit
<a href="#">RequestDeviation</a>	Получает максимально приемлемое отклонение от запрашиваемой цены
<a href="#">RequestType</a>	Получает тип ордера
<a href="#">RequestTypeDescription</a>	Получает тип ордера как строку
<a href="#">RequestTypeFilling</a>	Получает тип ордера по исполнению
<a href="#">RequestTypeFillingDescription</a>	Получает тип ордера по исполнению как строку
<a href="#">RequestTypeTime</a>	Получает тип ордера по истечению
<a href="#">RequestTypeTimeDescription</a>	Получает тип ордера по истечению как строку
<a href="#">RequestExpiration</a>	Получает срок истечения отложенного ордера
<a href="#">RequestComment</a>	Получает комментарий к ордеру
<a href="#">RequestPosition</a>	Получает тикет позиции
<a href="#">RequestPositionBy</a>	Получает тикет встречной позиции
<b>Доступ к результатам проверки последнего запроса</b>	
<a href="#">CheckResult</a>	Получает копию структуры результатов проверки последнего запроса
<a href="#">CheckResultRetcode</a>	Возвращает значение поля retcode структуры <a href="#">MqlTradeCheckResult</a> , заполненной при проверке правильности запроса
<a href="#">CheckResultRetcodeDescription</a>	Возвращает строковое описание поля retcode структуры <a href="#">MqlTradeCheckResult</a> , заполненной при проверке правильности запроса
<a href="#">CheckResultBalance</a>	Возвращает значение поля balance структуры <a href="#">MqlTradeCheckResult</a> , заполненной при проверке правильности запроса
<a href="#">CheckResultEquity</a>	Возвращает значение поля equity структуры <a href="#">MqlTradeCheckResult</a> , заполненной при проверке правильности запроса
<a href="#">CheckResultProfit</a>	Получает значение плавающей прибыли, которая будет после выполнения торговой операции

<a href="#">CheckResultMargin</a>	Возвращает значение поля margin структуры <a href="#">MqlTradeCheckResult</a> , заполненной при проверке правильности запроса
<a href="#">CheckResultMarginFree</a>	Возвращает значение поля margin_free структуры <a href="#">MqlTradeCheckResult</a> , заполненной при проверке правильности запроса
<a href="#">CheckResultMarginLevel</a>	Возвращает значение поля margin_level структуры <a href="#">MqlTradeCheckResult</a> , заполненной при проверке правильности запроса
<a href="#">CheckResultComment</a>	Возвращает значение поля comment структуры <a href="#">MqlTradeCheckResult</a> , заполненной при проверке правильности запроса
<b>Доступ к результатам исполнения последнего запроса</b>	
<a href="#">Result</a>	Получает копию структуры результатов исполнения последнего запроса
<a href="#">ResultRetcode</a>	Получает код результата выполнения запроса
<a href="#">ResultRetcodeDescription</a>	Получает код результата выполнения запроса как строку
<a href="#">ResultDeal</a>	Получает тикет сделки
<a href="#">ResultOrder</a>	Получает тикет ордера
<a href="#">ResultVolume</a>	Получает объем сделки или ордера
<a href="#">ResultPrice</a>	Получает цену, подтвержденную брокером
<a href="#">ResultBid</a>	Получает текущую цену предложения (реквоту)
<a href="#">ResultAsk</a>	Получает текущую цену спроса (реквоту)
<a href="#">ResultComment</a>	Получает комментарий брокера
<b>Вспомогательные методы</b>	
<a href="#">PrintRequest</a>	Выводит в журнал параметры последнего запроса
<a href="#">PrintResult</a>	Выводит в журнал результаты исполнения последнего запроса
<a href="#">FormatRequest</a>	Выводит в строку параметры последнего запроса
<a href="#">FormatRequestResult</a>	Выводит в строку результаты исполнения последнего запроса

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## LogLevel

Устанавливает уровень логирования сообщений.

```
void LogLevel(
    ENUM_LOG_LEVELS log_level      // уровень
)
```

### Параметры

*log\_level*  
[in] Новый уровень логирования.

### Возвращаемое значение

Нет.

### Примечание

LOG\_LEVEL\_NO и меньше, отключает вывод любых сообщений (автоматически устанавливается в режиме оптимизации). LOG\_LEVEL\_ERRORS включает вывод только сообщений об ошибках (значение по умолчанию). LOG\_LEVEL\_ALL и больше, включает вывод любых сообщений (автоматически устанавливается в режиме тестирования).

### ENUM\_LOG\_LEVELS

Идентификатор	Описание	Значение
LOG_LEVEL_NO	Вывод сообщений отключен	0
LOG_LEVEL_ERRORS	Выводятся только сообщения об ошибках	1
LOG_LEVEL_ALL	Выводятся все сообщения	2

## SetExpertMagicNumber

Устанавливает идентификатор эксперта

```
void SetExpertMagicNumber(  
    ulong magic // идентификатор  
)
```

### Параметры

*magic*

[in] Новый идентификатор эксперта.

### Возвращаемое значение

Нет.

## SetDeviationInPoints

Устанавливает допустимое проскальзывание.

```
void SetDeviationInPoints(
    ulong deviation          // проскальзывание
)
```

### Параметры

*deviation*

[in] Допустимое проскальзывание в пунктах.

### Возвращаемое значение

Нет.

## SetTypeFilling

Устанавливает тип ордера по исполнению.

```
void SetTypeFilling(  
    ENUM_ORDER_TYPE_FILLING filling // тип ордера по исполнению  
)
```

### Параметры

*filling*

[in] Тип ордера по исполнению из перечисления [ENUM\\_ORDER\\_TYPE\\_FILLING](#).

### Возвращаемое значение

Нет.

## SetTypeFillingBySymbol

Устанавливает тип ордера по [исполнению](#) согласно настройкам указанного символа.

```
bool SetTypeFillingBySymbol(
    const string    symbol      // имя финансового инструмента
)
```

### Параметры

*symbol*

[in] Имя символа, в котором [SYMBOL\\_FILLING\\_MODE](#) содержит допустимые политики исполнения ордеров.

### Возвращаемое значение

true - в случае удачного выполнения, false - если не удалось установить политику исполнения.

### Примечание

Если для символа одновременно разрешены политики исполнения [SYMBOL\\_FILLING\\_FOK](#) и [SYMBOL\\_FILLING\\_IOC](#), то для ордера будет установлено значение [ORDER\\_FILLING\\_FOK](#).

## SetAsyncMode

Устанавливает асинхронный режим проведения торговых операций.

```
void SetAsyncMode(  
    bool mode           // асинхронный режим торговых операций  
)
```

### Параметры

*mode*

[in] Статус использования асинхронного режима торговых операций.

### Возвращаемое значение

Нет.

### Примечание

Асинхронный режим позволяет проводить торговые операции без ожидания ответа торгового сервера (см. [OrderSendAsync](#)).

## SetMarginMode

Устанавливает режим расчета маржи в соответствии с настройками текущего счета.

```
void SetMarginMode()
```

### Возвращаемое значение

Нет.

### Примечание

Режим расчета маржи указывается в [ENUM\\_ACCOUNT\\_MARGIN\\_MODE](#).

## OrderOpen

Размещает отложенный ордер с заданными параметрами.

```
bool OrderOpen(
    const string      symbol,           // символ
    ENUM_ORDER_TYPE   order_type,        // тип ордера
    double            volume,            // объем ордера
    double            limit_price,       // цена стоплимита
    double            price,              // цена исполнения
    double            sl,                 // цена stop loss
    double            tp,                 // цена take profit
    ENUM_ORDER_TYPE_TIME type_time,     // тип по истечению
    datetime          expiration,        // истечение
    const string      comment=""        // комментарий
)
```

### Параметры

*symbol*

[in] Наименование торгового инструмента.

*order\_type*

[in] Тип торговой операции для ордера из перечисления [ENUM\\_ORDER\\_TYPE](#).

*volume*

[in] Запрашиваемый объем ордера.

*limit\_price*

[in] Цена, по которой будет выставлен отложенный Stop Limit ордер.

*price*

[in] Цена, по которой ордер должен быть исполнен.

*sl*

[in] Цена, по которой сработает Stop Loss (ограничение потерь).

*tp*

[in] Цена, по которой сработает Take Profit (фиксация прибыли).

*type\_time*

[in] Тип ордера по истечению из перечисления [ENUM\\_ORDER\\_TYPE\\_TIME](#).

*expiration*

[in] Срок истечения отложенного ордера.

*comment=""*

[in] Комментарий к ордеру.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

## Примечание

Успешное окончание работы метода OrderOpen(...) не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса (код возврата торгового сервера) вызовом метода [ResultRetcode\(\)](#), а также значение, возвращаемое методом [ResultOrder\(\)](#).

## OrderModify

Изменяет параметры отложенного ордера.

```
bool OrderModify(
    ulong           ticket,          // тикет ордера
    double          price,           // цена исполнения
    double          sl,              // цена stop loss
    double          tp,              // цена take profit
    ENUM_ORDER_TYPE_TIME type_time, // тип по истечению
    datetime        expiration,      // истечение
    double          stoplimit       // цена Limit ордера
)
```

### Параметры

*ticket*

[in] Тикет ордера.

*price*

[in] Новая цена, по которой ордер должен быть исполнен (либо предыдущее значение, если изменение не нужно).

*sl*

[in] Новая цена, по которой сработает Stop Loss (либо предыдущее значение, если изменение не нужно).

*tp*

[in] Новая цена, по которой сработает Take Profit (либо предыдущее значение, если изменение не нужно).

*type\_time*

[in] Новый тип ордера по истечению из перечисления [ENUM\\_ORDER\\_TYPE\\_TIME](#) (либо предыдущее значение, если изменение не нужно).

*expiration*

[in] Новый срок истечения отложенного ордера (либо предыдущее значение, если изменение не нужно).

*stoplimit*

[in] Новая цена, по которой будет выставлен Limit ордер при достижении ценой значения *price*. Указывается только для StopLimit ордеров.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода OrderModify(...) не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса (код возврата торгового сервера) вызовом метода [ResultRetcode\(\)](#).

## OrderDelete

Удаляет отложенный ордер.

```
bool OrderDelete(
    ulong ticket           // тикет ордера
)
```

### Параметры

*ticket*

[in] Тикет ордера.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода OrderDelete(...) не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса (код возврата торгового сервера) вызовом метода [ResultRetcode\(\)](#).

## PositionOpen

Открывает позицию с заданными параметрами по указанному символу.

```
bool PositionOpen(
    const string      symbol,           // символ
    ENUM_ORDER_TYPE  order_type,       // тип торговой операции для открытия позиции
    double            volume,          // объем позиции
    double            price,           // цена исполнения
    double            sl,              // цена Stop Loss
    double            tp,              // цена Take Profit
    const string      comment=""        // комментарий
)
```

### Параметры

*symbol*

[in] Наименование торгового инструмента, по которому предполагается открыть позицию.

*order\_type*

[in] Тип торговой операции для открытия позиции из перечисления [ENUM\\_ORDER\\_TYPE](#).

*volume*

[in] Запрашиваемый объем позиции.

*price*

[in] Цена, по которой позиция должна быть открыта.

*sl*

[in] Цена, по которой сработает Stop Loss (ограничение потерь).

*tp*

[in] Цена, по которой сработает Take Profit (фиксация прибыли).

*comment=""*

[in] Комментарий к позиции.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода PositionOpen(...) не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса (код возврата торгового сервера) вызовом метода [ResultRetcode\(\)](#), а также значение, возвращаемое методом [ResultDeal\(\)](#).

## PositionModify

Изменяет параметры позиции по указанному символу.

```
bool PositionModify(
    const string symbol,           // символ
    double sl,                    // цена Stop Loss
    double tp                      // цена Take Profit
)
```

Изменяет параметры позиции по указанному тикету.

```
bool PositionModify(
    const ulong ticket,           // тикет позиции
    double sl,                    // цена Stop Loss
    double tp                      // цена Take Profit
)
```

### Параметры

*symbol*

[in] Наименование торгового инструмента, по которому предполагается модифицировать позицию.

*ticket*

[in] Тикет позиции, которую предполагается модифицировать.

*sl*

[in] Новая цена, по которой сработает Stop Loss (либо, если изменение не нужно, предыдущее значение).

*tp*

[in] Новая цена, по которой сработает Take Profit (либо, если изменение не нужно, предыдущее значение).

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода PositionModify(...) не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса (код возврата торгового сервера) вызовом метода [ResultRetcode\(\)](#).

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций. В этом случае, PositionModify изменит позицию с наименьшим тикетом.



## PositionClose

Закрывает позицию по указанному символу.

```
bool PositionClose(
    const string symbol,           // символ
    ulong      deviation=ULONG_MAX // отклонение
)
```

Закрывает позицию с указанным тикетом.

```
bool PositionClose(
    const ulong ticket,           // тикет позиции
    ulong      deviation=ULONG_MAX // отклонение
)
```

### Параметры

*symbol*

[in] Наименование торгового инструмента, по которому предполагается закрыть позицию.

*ticket*

[in] Тикет закрываемой позиции.

*deviation=ULONG\_MAX*

[in] Максимальное отклонение от текущей цены (в пунктах).

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода PositionClose(...) не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса (код возврата торгового сервера) вызовом метода [ResultRetcode\(\)](#).

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций. В этом случае, PositionClose закроет позицию с наименьшим тикетом.

## PositionClosePartial

Закрывает часть позиции по указанному символу при "хеджинговом" учете.

```
bool PositionClosePartial(
    const string symbol,           // символ
    const double volume,          // объем
    ulong deviation=ULONG_MAX    // отклонение
)
```

Закрывает часть позиции с указанным тикетом при "хеджинговом" учете.

```
bool PositionClosePartial(
    const ulong ticket,           // тикет позиции
    const double volume,          // объем
    ulong deviation=ULONG_MAX    // отклонение
)
```

### Параметры

*symbol*

[in] Наименование торгового инструмента, по которому предполагается закрыть часть позиции. Если для частичного закрытия позиции указан символ (не тикет), то будет выбрана первая найденная позиция по данному символу, имеющая заданный MagicNumber ([идентификатор эксперта](#)). Поэтому в ряде случаев лучше использовать вариант PositionClosePartial() с указанием тикета позиции.

*volume*

[in] Объем, на который нужно уменьшить позицию. Если значение больше, чем объём частично закрываемой позиции, то произойдет полное закрытие позиции. Новая позиция в обратном направлении открыта не будет.

*ticket*

[in] Тикет закрываемой позиции.

*deviation=ULONG\_MAX*

[in] Максимальное отклонение от текущей цены (в пунктах).

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода PositionClosePartial(...) не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса (код возврата торгового сервера) вызовом метода [ResultRetcode\(\)](#).

При "неттинговом" учете позиций ([ACCOUNT\\_MARGIN\\_MODE RETAIL\\_NETTING](#) и [ACCOUNT\\_MARGIN\\_MODE\\_EXCHANGE](#)) по каждому [символу](#) в любой момент времени может быть открыта только одна [позиция](#), которая является результатом одной или более  [сделок](#). Не следует путать между собой позиции и действующие [отложенные ордера](#), которые также отображаются на вкладке "Торговля" в панели "Инструменты".

При независимом представлении позиций ([ACCOUNT\\_MARGIN\\_MODE\\_RETAIL\\_HEDGING](#)) по каждому символу одновременно может быть открыто несколько позиций. В этом случае, PositionClose закроет позицию с наименьшим тикетом.

## PositionCloseBy

Закрывает позицию с указанным тикетом позицией встречного направления.

```
bool PositionCloseBy(
    const ulong ticket,           // тикет позиции
    const ulong ticket_by         // тикет встречной позиции
)
```

### Параметры

*ticket*

[in] Тикет закрываемой позиции.

*ticket\_by*

[in] Тикет встречной позиции, которая используется для закрытия.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода PositionCloseBy(...) не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса (код возврата торгового сервера) вызовом метода [ResultRetcode\(\)](#).

## Buy

Открывает длинную позицию с заданными параметрами.

```
bool Buy(
    double      volume,           // объем позиции
    const string symbol=NULL,     // символ
    double      price=0.0,        // цена исполнения
    double      sl=0.0,           // цена Stop Loss
    double      tp=0.0,           // цена Take Profit
    const string comment=""       // комментарий
)
```

### Параметры

*volume*

[in] Запрашиваемый объем позиции.

*symbol=NULL*

[in] Наименование торгового инструмента, по которому предполагается открыть позицию. Если символ не указан, то позиция откроется по текущему инструменту.

*price=0.0*

[in] Цена, по которой позиция должна быть открыта. Если цена не указана, то позиция откроется по текущей рыночной цене Ask.

*sl=0.0*

[in] Цена, по которой сработает Stop Loss (ограничение потерь).

*tp=0.0*

[in] Цена, по которой сработает Take Profit (фиксация прибыли).

*comment=""*

[in] Комментарий к позиции.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода Buy(...) не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса ([код возврата](#) торгового сервера) вызовом метода [ResultRetcode\(\)](#), а также значение, возвращаемое методом [ResultDeal\(\)](#).

## Sell

Открывает короткую позицию с заданными параметрами.

```
bool Sell(
    double      volume,           // объем позиции
    const string symbol=NULL,     // символ
    double      price=0.0,        // цена исполнения
    double      sl=0.0,           // цена Stop Loss
    double      tp=0.0,           // цена Take Profit
    const string comment=""       // комментарий
)
```

### Параметры

*volume*

[in] Запрашиваемый объем позиции.

*symbol=NULL*

[in] Наименование торгового инструмента, по которому предполагается открыть позицию. Если символ не указан, то позиция откроется по текущему инструменту.

*price=0.0*

[in] Цена, по которой позиция должна быть открыта. Если цена не указана, то позиция откроется по текущей рыночной цене Bid.

*sl=0.0*

[in] Цена, по которой сработает Stop Loss (ограничение потерь).

*tp=0.0*

[in] Цена, по которой сработает Take Profit (фиксация прибыли).

*comment=""*

[in] Комментарий к позиции.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода Sell(...) не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса ([код возврата](#) торгового сервера) вызовом метода [ResultRetcode\(\)](#), а также значение, возвращаемое методом [ResultDeal\(\)](#).

## BuyLimit

Устанавливает отложенный ордер на покупку по цене лучше рыночной.

```
bool BuyLimit(
    double           volume,          // объем позиции
    double           price,           // цена исполнения
    const string     symbol=NULL,      // символ
    double           sl=0.0,          // цена Stop Loss
    double           tp=0.0,          // цена Take Profit
    ENUM_ORDER_TYPE_TIME type_time=ORDER_TIME_GTC, // тип истечения
    datetime         expiration=0,     // время истечения
    const string     comment=""       // комментарий
)
```

### Параметры

*volume*

[in] Запрашиваемый объем ордера.

*price*

[in] Цена, по которой предполагается установить ордер.

*symbol=NULL*

[in] Наименование торгового инструмента, по которому предполагается установить ордер. Если символ не указан, то ордер установится по текущему инструменту.

*sl=0.0*

[in] Цена, по которой сработает Stop Loss (ограничение потерь).

*tp=0.0*

[in] Цена, по которой сработает Take Profit (фиксация прибыли).

*type\_time=ORDER\_TIME\_GTC*

[in] Тип истечения срока действия ордера из перечисления [ENUM\\_ORDER\\_TYPE\\_TIME](#).

*expiration=0*

[in] Время истечения срока действия ордера (только для *type\_time=ORDER\_TIME\_SPECIFIED*).

*comment=""*

[in] Комментарий к ордеру.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода `BuyLimit(...)` не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса ([код возврата](#) торгового сервера) вызовом метода [ResultRetcode\(\)](#), а также значение, возвращаемое методом [ResultOrder\(\)](#).

## BuyStop

Устанавливает отложенный ордер на покупку по цене хуже рыночной.

```
bool BuyStop(
    double           volume,           // объем позиции
    double           price,            // цена исполнения
    const string     symbol=NULL,      // символ
    double           sl=0.0,           // цена Stop Loss
    double           tp=0.0,           // цена Take Profit
    ENUM_ORDER_TYPE_TIME type_time=ORDER_TIME_GTC, // тип истечения
    datetime         expiration=0,      // время истечения
    const string     comment=""       // комментарий
)
```

### Параметры

*volume*

[in] Запрашиваемый объем ордера.

*price*

[in] Цена, по которой предполагается установить ордер.

*symbol=NULL*

[in] Наименование торгового инструмента, по которому предполагается установить ордер. Если символ не указан, то ордер установится по текущему инструменту.

*sl=0.0*

[in] Цена, по которой сработает Stop Loss (ограничение потерь).

*tp=0.0*

[in] Цена, по которой сработает Take Profit (фиксация прибыли).

*type\_time=ORDER\_TIME\_GTC*

[in] Тип истечения срока действия ордера из перечисления [ENUM\\_ORDER\\_TYPE\\_TIME](#).

*expiration=0*

[in] Время истечения срока действия ордера (только для *type\_time=ORDER\_TIME\_SPECIFIED*).

*comment=""*

[in] Комментарий к ордеру.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода `BuyStop(...)` не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса ([код возврата](#) торгового сервера) вызовом метода [ResultRetcode\(\)](#), а также значение, возвращаемое методом [ResultOrder\(\)](#).



## SellLimit

Устанавливает отложенный ордер на продажу по цене лучше рыночной.

```
bool SellLimit(
    double           volume,           // объем позиции
    double           price,            // цена исполнения
    const string     symbol=NULL,      // символ
    double           sl=0.0,           // цена Stop Loss
    double           tp=0.0,           // цена Take Profit
    ENUM_ORDER_TYPE_TIME type_time=ORDER_TIME_GTC, // тип истечения
    datetime         expiration=0,      // время истечения
    const string     comment=""       // комментарий
)
```

### Параметры

*volume*

[in] Запрашиваемый объем ордера.

*price*

[in] Цена, по которой предполагается установить ордер.

*symbol=NULL*

[in] Наименование торгового инструмента, по которому предполагается установить ордер. Если символ не указан, то ордер установится по текущему инструменту.

*sl=0.0*

[in] Цена, по которой сработает Stop Loss (ограничение потерь).

*tp=0.0*

[in] Цена, по которой сработает Take Profit (фиксация прибыли).

*type\_time=ORDER\_TIME\_GTC*

[in] Тип истечения срока действия ордера из перечисления [ENUM\\_ORDER\\_TYPE\\_TIME](#).

*expiration=0*

[in] Время истечения срока действия ордера (только для *type\_time=ORDER\_TIME\_SPECIFIED*).

*comment=""*

[in] Комментарий к ордеру.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода `SellLimit(...)` не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса ([код возврата](#) торгового сервера) вызовом метода [ResultRetcode\(\)](#), а также значение, возвращаемое методом [ResultOrder\(\)](#).



## SellStop

Устанавливает отложенный ордер на продажу по цене хуже рыночной.

```
bool SellStop(
    double           volume,          // объем позиции
    double           price,           // цена исполнения
    const string     symbol=NULL,      // символ
    double           sl=0.0,          // цена Stop Loss
    double           tp=0.0,          // цена Take Profit
    ENUM_ORDER_TYPE_TIME type_time=ORDER_TIME_GTC, // тип истечения
    datetime         expiration=0,     // время истечения
    const string     comment=""       // комментарий
)
```

### Параметры

*volume*

[in] Запрашиваемый объем ордера.

*price*

[in] Цена, по которой предполагается установить ордер.

*symbol=NULL*

[in] Наименование торгового инструмента, по которому предполагается установить ордер. Если символ не указан, то ордер установится по текущему инструменту.

*sl=0.0*

[in] Цена, по которой сработает Stop Loss (ограничение потерь).

*tp=0.0*

[in] Цена, по которой сработает Take Profit (фиксация прибыли).

*type\_time=ORDER\_TIME\_GTC*

[in] Тип истечения срока действия ордера из перечисления [ENUM\\_ORDER\\_TYPE\\_TIME](#).

*expiration=0*

[in] Время истечения срока действия ордера (только для *type\_time=ORDER\_TIME\_SPECIFIED*).

*comment=""*

[in] Комментарий к ордеру.

### Возвращаемое значение

true - в случае успешной базовой проверки структур, иначе false.

### Примечание

Успешное окончание работы метода `SellStop(...)` не всегда означает успешное совершение торговой операции. Необходимо проверять результат выполнения торгового запроса ([код возврата](#) торгового сервера) вызовом метода [ResultRetcode\(\)](#), а также значение, возвращаемое методом [ResultOrder\(\)](#).



## Request

Получает копию структуры последнего запроса.

```
void Request(
    MqlTradeRequest& request          // ссылка
) const
```

### Параметры

*request*

[out] Ссылка на структуру для копирования.

### Возвращаемое значение

Нет.

## RequestAction

Получает тип торговой операции.

```
ENUM_TRADE_REQUEST_ACTIONS RequestAction() const
```

### Возвращаемое значение

Тип торговой операции, использованный в последнем запросе из перечисления [ENUM\\_TRADE\\_REQUEST\\_ACTIONS](#).

## RequestActionDescription

Получает тип торговой операции как строку.

```
string RequestActionDescription() const
```

### Возвращаемое значение

Тип торговой операции, использованный в последнем запросе, как строку.

## RequestMagic

Получает идентификатор эксперта.

```
ulong RequestMagic() const
```

### Возвращаемое значение

Идентификатор эксперта, использованный в последнем запросе.

## RequestOrder

Получает тикет ордера.

```
ulong RequestOrder() const
```

### Возвращаемое значение

Тикет ордера, использованный в последнем запросе.

## RequestSymbol

Получает имя торгового инструмента.

```
string RequestSymbol() const
```

### Возвращаемое значение

Имя торгового инструмента, по которому выставляется ордер, использованное в последнем запросе.

## RequestVolume

Получает запрашиваемый объем сделки в лотах.

```
double RequestVolume() const
```

### Возвращаемое значение

Запрашиваемый объем сделки в лотах, использованный в последнем запросе.

## RequestPrice

Получает цену, при достижении которой ордер должен быть исполнен.

```
double RequestPrice() const
```

### Возвращаемое значение

Цена, использованная в последнем запросе, при достижении которой ордер должен быть исполнен.

## RequestStopLimit

Получает цену, по которой будет выставлен отложенный Stop Limit ордер.

```
double RequestStopLimit() const
```

### Возвращаемое значение

Цена, использованная в последнем запросе, по которой будет выставлен отложенный Stop Limit ордер.

## RequestSL

Получает цену, по которой сработает Stop Loss.

```
double RequestSL() const
```

### Возвращаемое значение

Цена, использованная в последнем запросе, по которой сработает Stop Loss.

## RequestTP

Получает цену, по которой сработает Take Profit.

```
double RequestTP() const
```

### Возвращаемое значение

Цена, использованная в последнем запросе, по которой сработает Take Profit.

## RequestDeviation

Получает максимальное отклонение от запрашиваемой цены.

```
ulong RequestDeviation() const
```

### Возвращаемое значение

Максимально приемлемое отклонение от запрашиваемой цены, использованное в последнем запросе.

## RequestType

Получает тип ордера.

```
ENUM_ORDER_TYPE RequestType() const
```

### Возвращаемое значение

Тип ордера, использованный в последнем запросе из перечисления [ENUM\\_ORDER\\_TYPE](#).

## RequestTypeTimeDescription

Получает тип ордера по истечению как строку.

```
string RequestTypeTimeDescription() const
```

### Возвращаемое значение

Тип ордера по истечению, использованный в последнем запросе, как строку.

## RequestTypeFilling

Получает тип ордера по исполнению.

```
ENUM_ORDER_TYPE_FILLING RequestTypeFilling() const
```

### Возвращаемое значение

Тип ордера по исполнению (из перечисления [ENUM\\_ORDER\\_TYPE\\_FILLING](#)), использованный в последнем запросе.

## RequestTypeFillingDescription

Получает тип ордера по исполнению как строку.

```
string RequestTypeFillingDescription() const
```

### Возвращаемое значение

Тип ордера по исполнению, использованный в последнем запросе, как строку.

## RequestTypeTime

Получает тип ордера по истечению.

```
ENUM_ORDER_TYPE_TIME RequestTypeTime() const
```

### Возвращаемое значение

Тип ордера по истечению (из перечисления [ENUM\\_ORDER\\_TYPE\\_TIME](#)), использованный в последнем запросе.

## RequestTypeTimeDescription

Получает тип ордера по истечению как строку.

```
string RequestTypeTimeDescription() const
```

### Возвращаемое значение

Тип ордера по истечению, использованный в последнем запросе, как строку.

## RequestExpiration

Получает срок истечения отложенного ордера.

```
datetime RequestExpiration() const
```

### Возвращаемое значение

Срок истечения отложенного ордера, использованный в последнем запросе.

## RequestComment

Получает комментарий к ордеру.

```
string RequestComment() const
```

### Возвращаемое значение

Комментарий к ордеру, использованный в последнем запросе.

## RequestPosition

Получает тикет позиции.

```
ulong RequestPosition() const
```

### Возвращаемое значение

Тикет позиции, использованный в последнем запросе.

## RequestPositionBy

Получает тикет встречной позиции.

```
ulong RequestPositionBy() const
```

### Возвращаемое значение

Тикет встречной позиции, использованный в последнем запросе.

## Result

Получает копию структуры результата выполнения последнего запроса.

```
void Result(  
    MqlTradeResult& result // ссылка  
) const
```

### Параметры

*result*

[out] Ссылка на структуру типа [MqlTradeResult](#) для копирования.

### Возвращаемое значение

Нет.

## ResultRetcode

Получает код результата выполнения запроса.

```
uint ResultRetcode() const
```

### Возвращаемое значение

[Код результата](#) последнего запроса.

## ResultRetcodeDescription

Получает код результата выполнения запроса как строку.

```
string ResultRetcodeDescription() const
```

### Возвращаемое значение

Возвращает строку с описанием [результата последнего запроса](#).

## ResultDeal

Получает тикет сделки.

```
ulong ResultDeal() const
```

### Возвращаемое значение

Тикет сделки, если она совершена.

## ResultOrder

Получает тикет ордера.

```
ulong ResultOrder() const
```

### Возвращаемое значение

Тикет ордера, если он выставлен.

## ResultVolume

Получает объем сделки или ордера.

```
double ResultVolume() const
```

### Возвращаемое значение

Объем сделки или ордера, подтверждённый брокером.

## ResultPrice

Получает цену, подтвержденную брокером.

```
double ResultPrice() const
```

### Возвращаемое значение

Цена, подтвержденная брокером.

## ResultBid

Получает текущую цену предложения (реквоту).

```
double ResultBid() const
```

### Возвращаемое значение

Текущая рыночная цена предложения (реквота).

## ResultAsk

Получает текущую цену спроса (реквоту).

```
double ResultAsk() const
```

### Возвращаемое значение

Текущая рыночная цена спроса (цены реквота).

## ResultComment

Получает комментарий брокера.

```
string ResultComment() const
```

### Возвращаемое значение

Комментарий брокера к операции.

## CheckResult

Получает копию структуры результата проверки последнего запроса.

```
void CheckResult(
    MqlTradeCheckResult& check_result        // ссылка
) const
```

### Параметры

*check\_result*

[out] Ссылка на структуру типа [MqlTradeCheckResult](#) для копирования.

### Возвращаемое значение

Нет.

## CheckResultRetcode

Возвращает значение поля `retcode` структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

```
uint CheckResultRetcode() const
```

### Возвращаемое значение

Значение поля `retcode` (код ошибки) структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

## CheckResultRetcodeDescription

Возвращает строковое описание поля retcode структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

```
string ResultRetcodeDescription() const
```

### Возвращаемое значение

Строковое описание поля retcode (код ошибки) структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

## CheckResultBalance

Возвращает значение поля `balance` структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

```
double CheckResultBalance() const
```

### Возвращаемое значение

Значение поля `balance` (значение баланса, которое будет после выполнения торговой операции) структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

## CheckResultEquity

Возвращает значение поля equity структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

```
double CheckResultEquity() const
```

### Возвращаемое значение

Значение поля equity (значение собственных средств, которое будет после выполнения торговой операции) структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

## CheckResultProfit

Получает значение плавающей прибыли.

```
double CheckResultProfit() const
```

### Возвращаемое значение

Значение плавающей прибыли, которая будет после выполнения торговой операции.

## CheckResultMargin

Возвращает значение поля margin структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

```
double CheckResultMargin() const
```

### Возвращаемое значение

Значение поля margin (размер маржи необходимый для требуемой торговой операции) структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

## CheckResultMarginFree

Возвращает значение поля margin\_free структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

```
double CheckResultMarginFree() const
```

### Возвращаемое значение

Значение поля margin\_free (размер свободных собственных средств, которые останутся после выполнения требуемой торговой операции) структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

## CheckResultMarginLevel

Возвращает значение поля margin\_level структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

```
double CheckResultMarginLevel() const
```

### Возвращаемое значение

Значение поля margin\_level (уровень маржи, который установится после выполнения требуемой торговой операции) структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

## CheckResultComment

Возвращает значение поля comment структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

```
string CheckResultComment() const
```

### Возвращаемое значение

Значение поля comment (комментарий к коду ответа, описание ошибки) структуры [MqlTradeCheckResult](#), заполненной при проверке правильности запроса.

## PrintRequest

Выводит в журнал параметры последнего запроса.

```
void PrintRequest() const
```

### Возвращаемое значение

Нет.

## PrintResult

Выводит в журнал результаты исполнения последнего запроса.

```
void PrintResult() const
```

### Возвращаемое значение

Нет.

## FormatRequest

Выводит в строку параметры последнего запроса.

```
string FormatRequest(  
    string&           str,          // строка  
    const MqlTradeRequest& request   // запрос  
) const
```

### Параметры

*str*

[in] Ссылка на строку для размещения результата.

*request*

[in] Ссылка на структуру типа [MqlTradeRequest](#), содержащую информацию о последнем запросе.

### Возвращаемое значение

Нет.

## FormatRequestResult

Выводит в строку результаты исполнения последнего запроса.

```
string FormatRequestResult(
    string&           str,          // строка
    const MqlTradeRequest& request,   // запрос
    const MqlTradeResult& result      // результат
) const
```

### Параметры

*str*

[in] Ссылка на строку для размещения результата.

*request*

[in] Ссылка на структуру типа [MqlTradeRequest](#), содержащую информацию о последнем запросе.

*result*

[in] Ссылка на структуру типа [MqlTradeResult](#), содержащую информацию о результате исполнения последнего запроса.

### Возвращаемое значение

Нет.

## Класс CTerminalInfo

Класс CTerminalInfo является классом для упрощенного доступа к свойствам окружения mq5-программы.

### Описание

Класс CTerminalInfo обеспечивает упрощенный доступ к свойствам окружения mq5-программы.

### Декларация

```
class CTerminalInfo : public CObject
```

### Заголовок

```
#include <Trade\TerminalInfo.mqh>
```

### Иерархия наследования

CObject

CTerminalInfo

### Методы класса по группам

Методы доступа к параметрам типа integer клиента терминала	
<a href="#"><u>Build</u></a>	Получает номер билда запущенного терминала
<a href="#"><u>IsConnected</u></a>	Получает информацию о наличии подключения к торговому серверу
<a href="#"><u>IsDLLsAllowed</u></a>	Получает информацию о разрешении на использование DLL
<a href="#"><u>IsTradeAllowed</u></a>	Получает информацию о разрешении на торговлю
<a href="#"><u>IsEmailEnabled</u></a>	Получает информацию о разрешении на отправку писем с использованием SMTP-сервера и логина, указанных в настройках терминала
<a href="#"><u>IsFtpEnabled</u></a>	Получает информацию о разрешении на отправку отчетов по FTP для логина и FTP-сервера, указанного в настройках
<a href="#"><u>MaxBars</u></a>	Получает максимальное количество баров на графике
<a href="#"><u>CodePage</u></a>	Получает информацию о номере кодовой страницы языка, установленного в терминале

<a href="#"><u>CPUcores</u></a>	Получает информацию о количестве ядер процессора, установленных в системе
<a href="#"><u>MemoryPhysical</u></a>	Получает информацию о размере физической памяти в системе (в Mb)
<a href="#"><u>MemoryTotal</u></a>	Получает информацию о размере памяти, доступной процессу терминала/агента (в Mb)
<a href="#"><u>MemoryAvailable</u></a>	Получает информацию о размере свободной памяти, доступной процессу терминала/агента (в Mb)
<a href="#"><u>MemoryUsed</u></a>	Получает информацию о размере памяти, используемой процессом терминала/агента (в Mb)
<a href="#"><u>IsX64</u></a>	Получает информацию о типе клиентского терминала/агента
<a href="#"><u>OpenCLSupport</u></a>	Получает информацию о версии OpenCL, поддерживаемой видеокартой
<a href="#"><u>DiskSpace</u></a>	Получает информацию об объеме свободной памяти на диске
<b>Методы доступа к параметрам типа string клиентского терминала</b>	
<a href="#"><u>Language</u></a>	Получает информацию о языке терминала
<a href="#"><u>Name</u></a>	Получает информацию об наименовании терминала
<a href="#"><u>Company</u></a>	Получает информацию о наименовании компании
<a href="#"><u>Path</u></a>	Получает информацию о папке, в которой запущен терминал
<a href="#"><u>DataPath</u></a>	Получает информацию о папке, в которой хранятся данные терминала
<a href="#"><u>CommonDataPath</u></a>	Получает информацию о папке данных всех клиентских терминалов, установленных на компьютере
<b>Доступ к функциям API MQL5</b>	
<a href="#"><u>InfoInteger</u></a>	Получает информацию о параметре типа integer терминала
<a href="#"><u>InfoString</u></a>	Получает информацию о параметре типа string терминала

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## Build

Получает номер билда запущенного терминала.

```
int CBuild() const
```

### Возвращаемое значение

Номер билда запущенного клиентского терминала.

### Примечание

Номер билда терминала определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_BUILD](#)).

## IsConnected

Получает информацию о наличии подключения к торговому серверу.

```
bool IsConnected() const
```

### Возвращаемое значение

true, если терминал подключен к серверу, иначе false.

### Примечание

Подключение к торговому серверу определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_CONNECTED](#)).

## IsDLLsAllowed

Получает информацию о разрешении на использование DLL.

```
bool IsDLLsAllowed() const
```

### Возвращаемое значение

true, если использование DLL разрешено, иначе false.

### Примечание

Разрешение на использование DLL определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_DLLS\\_ALLOWED](#)).

## IsTradeAllowed

Получает информацию о разрешении на торговлю.

```
bool IsTradeAllowed() const
```

### Возвращаемое значение

true, если торговля разрешена, иначе false.

### Примечание

Разрешение на торговлю определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_TRADE\\_ALLOWED](#)).

## IsEmailEnabled

Получает информацию о разрешении на отправку писем с использованием SMTP-сервера и логина, указанных в настройках терминала.

```
bool IsEmailEnabled() const
```

### Возвращаемое значение

true, если отправка писем разрешена, иначе false.

### Примечание

Разрешение на отправку писем определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_EMAIL\\_ENABLED](#)).

## IsFtpEnabled

Получает информацию о разрешении на отправку отчетов по FTP для логина и FTP-сервера, указанного в настройках терминала.

```
bool IsFtpEnabled() const
```

### Возвращаемое значение

true, если отправка отчетов по FTP разрешена, иначе false.

### Примечание

Разрешение на отправку отчетов определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_FTP\\_ENABLED](#)).

## MaxBars

Получает максимальное количество баров на графике, указанное в настройках терминала.

```
int MaxBars() const
```

### Возвращаемое значение

Максимальное количество баров на графике.

### Примечание

Максимальное количество баров на графике определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_MAXBARS](#)).

## CodePage

Получает информацию о номере кодовой страницы языка, установленного в терминале.

```
int CodePage() const
```

### Возвращаемое значение

Номер кодовой страницы языка, установленного в клиентском терминале.

### Примечание

Номер кодовой страницы языка определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_CODEPAGE](#)).

## CPUCores

Получает информацию о количестве ядер процессора, установленных в системе.

```
int CPUCores() const
```

### Возвращаемое значение

Количество ядер процессора.

### Примечание

Количество ядер процессора определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_CPU\\_CORES](#)).

## MemoryPhysical

Получает информацию о размере физической памяти в системе (в Mb).

```
int MemoryPhysical() const
```

### Возвращаемое значение

Размер физической памяти.

### Примечание

Размер физической памяти определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_MEMORY\\_PHYSICAL](#)).

## MemoryTotal

Получает информацию о размере памяти, доступной процессу терминала/агента (в Mb).

```
int MemoryTotal() const
```

### Возвращаемое значение

Размер памяти, доступный процессу терминала (агента).

### Примечание

Размер памяти, доступный процессу терминала (агента), определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_MEMORY\\_TOTAL](#)).

## MemoryAvailable

Получает информацию о размере свободной памяти, доступной процессу терминала/агента (в Mb).

```
int MemoryTotal() const
```

### Возвращаемое значение

Размер свободной памяти, доступный процессу терминала (агента).

### Примечание

Размер свободной памяти, доступный процессу терминала (агента), определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_MEMORY\\_TOTAL](#)).

## MemoryUsed

Получает информацию о размере памяти, используемой процессом терминала/агента (в Mb).

```
int MemoryUsed() const
```

### Возвращаемое значение

Размер памяти, используемой процессом терминала (агента).

### Примечание

Размер памяти, используемой процессом терминала (агента), определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_MEMORY\\_USED](#)).

## IsX64

Получает информацию о типе клиентского терминала/агента.

```
bool IsX64() const
```

### Возвращаемое значение

true, если используется 64-битная версия терминала, иначе false.

### Примечание

Тип терминала определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_X64](#)).

## OpenCLSupport

Получает информацию о версии OpenCL, поддерживаемой видеокартой.

```
int OpenCLSupport() const
```

### Возвращаемое значение

Версия OpenCL в виде 0x00010002 = "1.2". Значение 0 означает, что OpenCL не поддерживается.

### Примечание

Версия OpenCL определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_OPENCL\\_SUPPORT](#)).

## DiskSpace

Получает информацию об объеме свободной памяти на диске для папки MQL5\Files терминала/агента (в Mb).

```
int MDiskSpace() const
```

### Возвращаемое значение

Размер свободной дисковой памяти.

### Примечание

Размер свободной памяти на диске определяется при помощи функции [TerminalInfoInteger\(\)](#) (свойство [TERMINAL\\_DISK\\_SPACE](#)).

## Language

Получает информацию о языке клиентского терминала.

```
string Language() const
```

### Возвращаемое значение

Язык клиентского терминала.

### Примечание

Язык терминала определяется при помощи функции [TerminalInfoString\(\)](#) (свойство [TERMINAL\\_LANGUAGE](#)).

## Name

Получает информацию об наименовании клиентского терминала.

```
string Name() const
```

### Возвращаемое значение

Наименование клиентского терминала.

### Примечание

Наименование терминала определяется при помощи функции [TerminalInfoString\(\)](#) (свойство [TERMINAL\\_NAME](#)).

## Company

Получает информацию о наименовании компании.

```
string Company() const
```

### Возвращаемое значение

Наименование компании.

### Примечание

Наименование компании определяется при помощи функции [TerminalInfoString\(\)](#) (свойство [TERMINAL\\_COMPANY](#)).

## Path

Получает информацию о папке, в которой запущен терминал.

```
string Path() const
```

### Возвращаемое значение

Папка клиентского терминала.

### Примечание

Папка клиентского терминала определяется при помощи функции [TerminalInfoString\(\)](#) (свойство [TERMINAL\\_PATH](#)).

## DataPath

Получает информацию о папке, в которой хранятся данные терминала.

```
string DataPath() const
```

### Возвращаемое значение

Папка данных клиентского терминала.

### Примечание

Папка данных клиентского терминала определяется при помощи функции [TerminalInfoString\(\)](#) (свойство [TERMINAL\\_DATA\\_PATH](#)).

## CommonDataPath

Получает информацию об общей папке данных всех клиентских терминалов, установленных на компьютере.

```
string CommonDataPath() const
```

### Возвращаемое значение

Общая папка данных всех установленных терминалов.

### Примечание

Общая папка данных всех установленных терминалов определяется при помощи функции [TerminalInfoString\(\)](#) (свойство [COMMON\\_DATA\\_PATH](#)).

## InfoInteger

Возвращает значение соответствующего integer-свойства окружения mq5-программы.

```
int InfoInteger(
    int property_id           // идентификатор свойства
);
```

### Параметры

*property\_id*  
[in] Идентификатор свойства. Может быть одним из значений перечисления [ENUM\\_TERMINAL\\_INFO\\_INTEGER](#).

### Возвращаемое значение

Значение типа int.

### Примечание

Значение свойства определяется при помощи функции [TerminalInfoInteger\(\)](#).

## InfoString

Функция возвращает значение соответствующего свойства окружения MQL5-программы. Свойство должно быть типа string.

```
string InfoString(  
    int property_id // идентификатор свойства  
) ;
```

### Параметры

*property\_id*  
[in] Идентификатор свойства. Может быть одним из значений перечисления [ENUM\\_TERMINAL\\_INFO\\_STRING](#).

### Возвращаемое значение

Значение типа string.

### Примечание

Значение свойства определяется при помощи функции [TerminalInfoString\(\)](#).

## Набор классов для создания и проверки торговых стратегий

Этот раздел содержит технические детали работы с классами для создания и проверки торговых стратегий и описание соответствующих компонентов стандартной библиотеки MQL5.

Использование классов для создания и проверки торговых стратегий позволит сэкономить время при разработке (а особенно, проверке) торговых стратегий.

Стандартная библиотека MQL5 (в части набора классов для создания и проверки торговых стратегий) размещается в рабочем каталоге терминала в папке `Include\Expert`.

Базовые классы экспертов	Описание
<a href="#">CExpertBase</a>	Базовый класс для классов торговых стратегий
<a href="#">CExpert</a>	Базовый класс торговой стратегии
<a href="#">CExpertSignal</a>	Базовый класс для создания генераторов торговых сигналов
<a href="#">CExpertTrailing</a>	Базовый класс сопровождения открытых позиций
<a href="#">CExpertMoney</a>	Базовый класс для реализации алгоритмов управления капиталом и рисками

Классы торговых сигналов	Описание
<a href="#">CSignalAC</a>	Модуль сигналов на основе рыночных моделей индикатора Accelerator Oscillator.
<a href="#">CSignalAMA</a>	Модуль сигналов на основе рыночных моделей индикатора Adaptive Moving Average.
<a href="#">CSignalAO</a>	Модуль сигналов на основе рыночных моделей индикатора Awesome Oscillator.
<a href="#">CSignalBearsPower</a>	Модуль сигналов на основе рыночных моделей осциллятора Bears Power.
<a href="#">CSignalBullsPower</a>	Модуль сигналов на основе рыночных моделей осциллятора Bulls Power.
<a href="#">CSignalCCI</a>	Модуль сигналов на основе рыночных моделей осциллятора Commodity Channel Index.
<a href="#">CSignalDeM</a>	Модуль сигналов на основе рыночных моделей осциллятора DeMarker.

Классы торговых сигналов	Описание
<a href="#">CSignalDEMA</a>	Модуль сигналов на основе рыночных моделей индикатора Double Exponential Moving Average.
<a href="#">CSignalEnvelopes</a>	Модуль сигналов на основе рыночных моделей индикатора Envelopes.
<a href="#">CSignalFrAMA</a>	Модуль сигналов на основе рыночных моделей индикатора Fractal Adaptive Moving Average.
<a href="#">CSignalITF</a>	Модуль фильтрации сигналов по времени.
<a href="#">CSignalMACD</a>	Модуль сигналов на основе рыночных моделей осциллятора MACD.
<a href="#">CSignalMA</a>	Модуль сигналов на основе рыночных моделей индикатора Moving Average.
<a href="#">CSignalSAR</a>	Модуль сигналов на основе рыночных моделей индикатора Parabolic SAR.
<a href="#">CSignalRSI</a>	Модуль сигналов на основе рыночных моделей осциллятора Relative Strength Index.
<a href="#">CSignalRVI</a>	Модуль сигналов на основе рыночных моделей осциллятора Relative Vigor Index.
<a href="#">CSignalStoch</a>	Модуль сигналов на основе рыночных моделей осциллятора Stochastic.
<a href="#">CSignalTRIX</a>	Модуль сигналов на основе рыночных моделей осциллятора Triple Exponential Average.
<a href="#">CSignalTEMA</a>	Модуль сигналов на основе рыночных моделей индикатора Triple Exponential Moving Average.
<a href="#">CSignalWPR</a>	Модуль сигналов на основе рыночных моделей осциллятора Williams Percent Range.

Классы сопровождения открытых позиций	Описание
<a href="#">CTailingFixedPips</a>	Класс реализации алгоритма сопровождения открытых позиций на фиксированном "расстоянии"
<a href="#">CTailingMA</a>	Класс реализации алгоритма сопровождения открытых позиций по значениям скользящей средней
<a href="#">CTailingNone</a>	Класс-заглушка алгоритмов сопровождения открытых позиций

[CTrailingPSAR](#)

Класс реализации алгоритма сопровождения открытых позиций по значениям индикатора Parabolic SAR

Классы управления капиталом и рисками	Описание
<u><a href="#">CMoneyFixedLot</a></u>	Класс реализации алгоритма для входа в рынок фиксированным лотом, заданным при настройке
<u><a href="#">CMoneyFixedMargin</a></u>	Класс реализации алгоритма для входа в рынок фиксированным уровнем маржи, заданным при настройке
<u><a href="#">CMoneyFixedRisk</a></u>	Класс реализации алгоритма для входа в рынок фиксированным уровнем риска, заданным при настройке
<u><a href="#">CMoneyNone</a></u>	Класс реализации алгоритма для входа в рынок минимальным лотом
<u><a href="#">CMoneySizeOptimized</a></u>	Класс реализации алгоритма для входа в рынок с объемом, определяемом результатами предыдущих сделок

## Базовые классы для создания экспертов

Этот раздел содержит технические детали работы с классами для создания и проверки торговых стратегий и описание соответствующих компонентов стандартной библиотеки MQL5.

Использование классов для создания и проверки торговых стратегий позволит сэкономить время при разработке (а особенно, проверке) торговых стратегий.

Стандартная библиотека MQL5 (в части набора классов для создания и проверки торговых стратегий) размещается в рабочем каталоге терминала в папке `Include\Expert`.

Класс	Описание
<a href="#">CExpertBase</a>	Базовый класс для классов торговых стратегий
<a href="#">CExpert</a>	Базовый класс для реализации торговых стратегий
<a href="#">CExpertSignal</a>	Базовый класс для создания генераторов торговых сигналов
<a href="#">CExpertTrailing</a>	Базовый класс для реализации алгоритмов сопровождения открытых позиций
<a href="#">CExpertMoney</a>	Базовый класс для реализации алгоритмов управления капиталом и рисками

## Класс CExpertBase

Класс CExpertBase является базовым классом для класса [CExpert](#) и всех вспомогательных классов торговых стратегий.

### Описание

Класс CExpertBase предоставляет набор данных и методов, общих для всех составных частей эксперта.

### Декларация

```
class CExpertBase : public CObject
```

### Заголовок

```
#include <Expert\ExpertBase.mqh>
```

### Иерархия наследования

[CObject](#)

CExpertBase

### Прямые потомки

[CExpert](#), [CExpertMoney](#), [CExpertSignal](#), [CExpertTrailing](#)

### Методы класса по группам

#### Публичные методы:

Инициализация	
<a href="#">virtual Init</a>	Инициализирует объект
<a href="#">virtual ValidationSettings</a>	Проверяет корректность настроек
Установка параметров	
<a href="#">Symbol</a>	Устанавливает рабочий инструмент объекта
<a href="#">Period</a>	Устанавливает рабочий таймфрейм объекта
<a href="#">Magic</a>	Устанавливает идентификатор (magic) советника
Методы создания индикаторов и таймсерий	
<a href="#">virtual SetPriceSeries</a>	Устанавливает указатели на внешние ценовые объекты-таймсерии.
<a href="#">virtual SetOtherSeries</a>	Устанавливает указатели на внешние неценовые объекты-таймсерии

<code>virtual InitIndicators</code>	Инициализирует необходимые индикаторы и таймсерии
<b>Доступ к защищенным данным</b>	
<code>InitPhase</code>	Получает текущую фазу инициализации объекта
<code>TrendType</code>	Устанавливает идентификатор тренда
<code>UsedSeries</code>	Получает список используемых таймсерий
<code>EveryTick</code>	Устанавливает флаг анализа текущего бара
<b>Доступ к таймсериям</b>	
<code>Open</code>	Получает значение элемента таймсерии Open по указанному индексу
<code>High</code>	Получает значение элемента таймсерии High по указанному индексу
<code>Low</code>	Получает значение элемента таймсерии Low по указанному индексу
<code>Close</code>	Получает значение элемента таймсерии Close по указанному индексу
<code>Spread</code>	Получает значение элемента таймсерии Spread по указанному индексу
<code>Time</code>	Получает значение элемента таймсерии Time по указанному индексу
<code>TickVolume</code>	Получает значение элемента таймсерии TickVolume по указанному индексу
<code>RealVolume</code>	Получает значение элемента таймсерии RealVolume по указанному индексу

### Защищенные методы:

Инициализация таймсерий	
<code>InitOpen</code>	Инициализирует таймсерию Open
<code>InitHigh</code>	Инициализирует таймсерию High
<code>InitLow</code>	Инициализирует таймсерию Low
<code>InitClose</code>	Инициализирует таймсерию Close
<code>InitSpread</code>	Инициализирует таймсерию Spread
<code>InitTime</code>	Инициализирует таймсерию Time
<code>InitTickVolume</code>	Инициализирует таймсерию TickVolume

<a href="#">InitRealVolume</a>	Инициализирует таймсерию RealVolume
<b>Служебные методы</b>	
virtual <a href="#">PriceLevelUnit</a>	Получает единицу измерения ценовых уровней
virtual <a href="#">startIndex</a>	Получает номер индекса первого анализируемого бара
virtual <a href="#">CompareMagic</a>	Сравнивает значение идентификатора (magic) с указанным значением

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## InitPhase

Получает текущую фазу инициализации объекта.

```
ENUM_INIT_PHASE InitPhase()
```

### Возвращаемое значение

Текущая фаза инициализации объекта.

### Примечание

Инициализация объекта состоит из нескольких фаз (этапов). Назовем их условно:

1. Фаза начальной инициализации.

- начало - после завершения работы конструктора
- завершение - после удачного завершения работы метода [Init\(...\)](#)
- разрешено - вызов метода [Init\(...\)](#)
- запрещено - вызов любых других методов инициализации и метода [ValidationSettings\(...\)](#)

2. Фаза настройки параметров. Во время этой фазы необходимо установить все параметры объекта, которые будут использоваться для создания индикаторов.

- начало - после удачного завершения работы метода [Init\(...\)](#)
- завершение - после удачного завершения работы метода [ValidationSettings\(...\)](#)
- разрешено - вызовы методов [Symbol\(...\)](#) и [Period\(...\)](#)
  - запрещено - вызовы методов [Init\(...\)](#), [SetPriceSeries\(...\)](#), [SetOtherSeries\(...\)](#) и [InitIndicators\(...\)](#)

3. Фаза проверки параметров.

- начало - после удачного завершения работы метода [ValidationSettings\(\)](#)
- завершение - после удачного завершения работы метода [InitIndicators\(...\)](#)
- разрешено - вызовы методов [Symbol\(...\)](#), [Period\(...\)](#) и [InitIndicators\(...\)](#)
- запрещено - вызов любых других методов инициализации

4. Фаза завершения инициализации.

- начало - после удачного завершения работы метода [InitIndicators\(...\)](#)
- запрещено - вызов методов инициализации

## TrendType

Устанавливает идентификатор тренда.

```
void TrendType(
    M_TYPE_TREND    value          // значение
)
```

### Параметры

*value*

[in] Новое значение идентификатора тренда.

### Возвращаемое значение

Нет.

## UsedSeries

Получает список используемых таймсерий.

```
int UsedSeries()
```

### Возвращаемое значение

Список используемых таймсерий в виде битовой карты.

### Примечание

Если бит установлен, соответствующая таймсерия используется, если сброшен - нет.

Соответствие бит-таймсерия:

- бит 0 - таймсерия Open,
- бит 1 - таймсерия High,
- бит 2 - таймсерия Low,
- бит 3 - таймсерия Close,
- бит 4 - таймсерия Spread,
- бит 5 - таймсерия Time,
- бит 6 - таймсерия TickVolume,
- бит 7 - таймсерия RealVolume.

## EveryTick

Устанавливает флаг анализа текущего бара.

```
void EveryTick(
    bool    value          // флаг
)
```

### Параметры

*value*

[in] Новое значение флага.

### Возвращаемое значение

Нет.

### Примечание

Если флаг установлен, обрабатывается каждое изменение цены (тик) по рабочему инструменту.

Если флаг сброшен, обработка производится только по факту формирования нового бара по рабочему инструменту на рабочем таймфрейме.

## Open

Получает значение элемента таймсерии Open по указанному индексу.

```
double Open(
    int     ind          // индекс
)
```

### Параметры

*ind*

[in] Индекс элемента таймсерии.

### Возвращаемое значение

Значение элемента таймсерии Open по указанному индексу случае успешного завершения, иначе EMPTY\_VALUE.

### Примечание

Значение EMPTY\_VALUE может быть получено в двух случаях:

1. Таймсерия не используется (не был установлен соответствующий бит использования).
2. Индекс элемента вышел за пределы подготовленных данных таймсерии.

## High

Получает значение элемента таймсерии High по указанному индексу.

```
double  High(
    int     ind          // индекс
)
```

### Параметры

*ind*

[in] Индекс элемента таймсерии.

### Возвращаемое значение

Значение элемента таймсерии High по указанному индексу случае успешного завершения, иначе EMPTY\_VALUE.

### Примечание

Значение EMPTY\_VALUE может быть получено в двух случаях:

1. Таймсерия не используется (не был установлен соответствующий бит использования).
2. Индекс элемента вышел за пределы подготовленных данных таймсерии.

## Low

Получает значение элемента таймсерии Low по указанному индексу.

```
double Low(
    int     ind          // индекс
)
```

### Параметры

*ind*

[in] Индекс элемента таймсерии.

### Возвращаемое значение

Значение элемента таймсерии Low по указанному индексу случае успешного завершения, иначе EMPTY\_VALUE.

### Примечание

Значение EMPTY\_VALUE может быть получено в двух случаях:

1. Таймсерия не используется (не был установлен соответствующий бит использования).
2. Индекс элемента вышел за пределы подготовленных данных таймсерии.

## Close

Получает значение элемента таймсерии Close по указанному индексу.

```
double Close(
    int     ind          // индекс
)
```

### Параметры

*ind*

[in] Индекс элемента таймсерии.

### Возвращаемое значение

Значение элемента таймсерии Close по указанному индексу случае успешного завершения, иначе EMPTY\_VALUE.

### Примечание

Значение EMPTY\_VALUE может быть получено в двух случаях:

1. Таймсерия не используется (не был установлен соответствующий бит использования).
2. Индекс элемента вышел за пределы подготовленных данных таймсерии.

## Spread

Получает значение элемента таймсерии Spread по указанному индексу.

```
double Spread(
    int     ind          // индекс
)
```

### Параметры

*ind*

[in] Индекс элемента таймсерии.

### Возвращаемое значение

Значение элемента таймсерии Spread по указанному индексу случае успешного завершения, иначе EMPTY\_VALUE.

### Примечание

Значение EMPTY\_VALUE может быть получено в двух случаях:

1. Таймсерия не используется (не был установлен соответствующий бит использования).
2. Индекс элемента вышел за пределы подготовленных данных таймсерии.

## Time

Получает значение элемента таймсерии Time по указанному индексу.

```
datetime Time(
    int     ind          // индекс
)
```

### Параметры

*ind*

[in] Индекс элемента таймсерии.

### Возвращаемое значение

Значение элемента таймсерии Time по указанному индексу случае успешного завершения, иначе EMPTY\_VALUE.

### Примечание

Значение EMPTY\_VALUE может быть получено в двух случаях:

1. Таймсерия не используется (не был установлен соответствующий бит использования).
2. Индекс элемента вышел за пределы подготовленных данных таймсерии.

## TickVolume

Получает значение элемента таймсерии TickVolume по указанному индексу.

```
long TickVolume(
    int     ind          // индекс
)
```

### Параметры

*ind*

[in] Индекс элемента таймсерии.

### Возвращаемое значение

Значение элемента таймсерии TickVolume по указанному индексу случае успешного завершения, иначе EMPTY\_VALUE.

### Примечание

Значение EMPTY\_VALUE может быть получено в двух случаях:

1. Таймсерия не используется (не был установлен соответствующий бит использования).
2. Индекс элемента вышел за пределы подготовленных данных таймсерии.

## RealVolume

Получает значение элемента таймсерии RealVolume по указанному индексу.

```
long  RealVolume(
    int   ind          // индекс
)
```

### Параметры

*ind*

[in] Индекс элемента таймсерии.

### Возвращаемое значение

Значение элемента таймсерии RealVolume по указанному индексу случае успешного завершения, иначе EMPTY\_VALUE.

### Примечание

Значение EMPTY\_VALUE может быть получено в двух случаях:

1. Таймсерия не используется (не был установлен соответствующий бит использования).
2. Индекс элемента вышел за пределы подготовленных данных таймсерии.

## Init

Инициализирует объект.

```
bool Init(
    CSymbolInfo     symbol,    //
    ENUM_TIMEFRAMES period,   //
    double          point     //
)
```

### Параметры

*symbol*

[in] Указатель на объект [CSymbolInfo](#) для доступа к информации о рабочем инструменте.

*period*

[in] Рабочий таймфрейм из перечисления [ENUM\\_TIMEFRAMES](#).

*point*

[in] "Вес" 2/4 знакового пункта.

### Возвращаемое значение

true - в случае успешного завершения, иначе false.

## Symbol

Устанавливает рабочий инструмент объекта.

```
bool Symbol(
    string     name      // символ
)
```

### Параметры

*name*

[in] Наименование рабочего инструмента.

### Возвращаемое значение

true-в случае успешного завершения, иначе false.

### Примечание

Установка рабочего инструмента необходима в том случае если объект использует инструмент, отличный от установленного при начальной инициализации.

## Period

Устанавливает рабочий таймфрейм объекта.

```
bool Period(
    ENUM_TIMEFRAMES value        // таймфрейм
)
```

### Параметры

*value*

[in] Рабочий таймфрейм.

### Возвращаемое значение

true-в случае успешного завершения, иначе false.

### Примечание

Установка рабочего таймфрейма необходима в том случае, если объект использует таймфрейм, отличный от установленного при начальной инициализации.

## Magic

Устанавливает идентификатор (magic) советника.

```
void Magic(  
    ulong value      // идентификатор  
)
```

### Параметры

*value*

[in] Идентификатор советника.

### Возвращаемое значение

Нет.

## ValidationSettings

Проверяет корректность настроек.

```
virtual bool ValidationSettings()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## SetPriceSeries

Устанавливает указатели на внешние ценовые объекты-таймсерии.

```
virtual bool SetPriceSeries(
    CiOpen*   open,          // указатель
    CiHigh*   high,          // указатель
    CiLow*    low,           // указатель
    CiClose*  close          // указатель
)
```

### Параметры

*open*

[in] Указатель для доступа к таймсерии Open.

*high*

[in] Указатель для доступа к таймсерии High.

*low*

[in] Указатель для доступа к таймсерии Low.

*close*

[in] Указатель для доступа к таймсерии Close.

### Возвращаемое значение

true-в случае успешного завершения, иначе false.

### Примечание

Установка указателей на объекты доступа к ценовым таймсериям необходима в том случае, если объект использует инструмент и таймфрейм (установленные при начальной инициализации) и ценовые таймсерии, необходимые для дальнейшей работы.

## SetOtherSeries

Устанавливает указатели на внешние неценовые объекты-таймсерии.

```
virtual bool SetOtherSeries(
    CiSpread*      spread,          // указатель
    CiTime*         time,           // указатель
    CiTickVolume*   tick_volume,     // указатель
    CiRealVolume*   real_volume    // указатель
)
```

### Параметры

*spread*

[in] Указатель для доступа к таймсерии Spread.

*time*

[in] Указатель для доступа к таймсерии Time.

*tick\_volume*

[in] Указатель для доступа к таймсерии TickVolume.

*real\_volume*

[in] Указатель для доступа к таймсерии RealVolume.

### Возвращаемое значение

true-в случае успешного завершения, иначе false.

### Примечание

Установка указателей на объекты доступа к неценовым таймсериям необходима в том случае, если объект использует инструмент и таймфрейм (установленные при начальной инициализации) и неценовые таймсерии, необходимые для дальнейшей работы.

## InitIndicators

Инициализирует необходимые индикаторы и таймсерии.

```
virtual bool InitIndicators(
    CIndicators* indicators=NULL      // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true-в случае успешного завершения, иначе false.

### Примечание

Необходимые таймсерии инициализируются в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации.

## InitOpen

Инициализирует таймсерию Open.

```
bool InitOpen(
    CIIndicators* indicators // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true-в случае успешного завершения, иначе false.

### Примечание

Таймсерия Open инициализируется в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации и таймсерия необходима для дальнейшей работы.

## InitHigh

Инициализирует таймсерию High.

```
bool InitHigh(
    CIndicators* indicators // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true - в случае успешного завершения, иначе false.

### Примечание

Таймсерия High инициализируется в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации и таймсерия необходима для дальнейшей работы.

## InitLow

Инициализирует таймсерию Low.

```
bool InitLow(
    CIndicators* indicators // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true - в случае успешного завершения, иначе false.

### Примечание

Таймсерия Low инициализируется в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации и таймсерия необходима для дальнейшей работы.

## InitClose

Инициализирует таймсерию Close.

```
bool InitClose(
    CIndicators* indicators // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true - в случае успешного завершения, иначе false.

### Примечание

Таймсерия Close инициализируется в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации и таймсерия необходима для дальнейшей работы.

## InitSpread

Инициализирует таймсерию Spread.

```
bool InitSpread(
    CIIndicators* indicators // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true - в случае успешного завершения, иначе false.

### Примечание

Таймсерия Spread инициализируется в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации и таймсерия необходима для дальнейшей работы.

## InitTime

Инициализирует таймсерию Time.

```
bool InitTime(
    CIndicators* indicators // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true-в случае успешного завершения, иначе false.

### Примечание

Таймсерия Time инициализируется в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации и таймсерия необходима для дальнейшей работы.

## InitTickVolume

Инициализирует таймсерию TickVolume.

```
bool InitTickVolume(
    CIndicators* indicators // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true-в случае успешного завершения, иначе false.

### Примечание

Таймсерия TickVolume инициализируется в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации и таймсерия необходима для дальнейшей работы.

## InitRealVolume

Инициализирует таймсерию RealVolume.

```
bool InitRealVolume(
    CIndicators* indicators // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true-в случае успешного завершения, иначе false.

### Примечание

Таймсерия RealVolume инициализируется в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации и таймсерия необходима для дальнейшей работы.

## PriceLevelUnit

Получает единицу измерения ценовых уровней.

```
virtual double PriceLevelUnit()
```

### Возвращаемое значение

Единица измерения ценовых уровней.

### Примечание

В базовом классе возвращается "вес" 2/4 знакового пункта.

## StartIndex

Получает номер индекса первого анализируемого бара.

```
virtual int StartIndex()
```

### Возвращаемое значение

Номер индекса первого анализируемого бара.

### Примечание

Если флаг анализа текущего бара установлен, возвращается 0 (анализ будет производится с текущего бара). Если флаг анализа текущего бара сброшен, возвращается 1 (анализ будет производится с последнего сформировавшегося бара).

## CompareMagic

Сравнивает значение идентификатора (*magic*) с указанным значением.

```
virtual bool CompareMagic(  
    ulong magic // идентификатор  
)
```

### Параметры

*magic*

[in] Значение идентификатора для сравнения.

### Возвращаемое значение

true-в случае, если идентификатор совпадает с указанным, иначе false.

## Класс CExpert

Класс CExpert является базовым классом для реализации торговых стратегий.

В него уже заложены элементарные "навыки" торговли. Класс имеет встроенные механизмы для работы с таймсериями и индикаторами и набор виртуальных методов, определяющих торговую стратегию.

Для того чтобы ваш эксперт заработал по-другому, нужно:

1. Определиться с алгоритмами стратегии;
2. Создать свой класс, унаследовав его от CExpert;
3. Переопределить в своем классе виртуальные методы базового, заложив в них соответствующие алгоритмы.

### Описание

Класс CExpert предоставляет набор виртуальных методов для реализации торговых стратегий.

### Примечание

Позиция определяется принадлежащей эксперту и управляемой экспертом по паре свойств m\_symbol и m\_magic. В режиме "хеджинга" на счете одновременно может существовать несколько позиций по одному и тому же символу, поэтому особое внимание следует уделять значению m\_magic.

### Декларация

```
class CExpert : public CExpertBase
```

### Заголовок

```
#include <Expert\Expert.mqh>
```

### Иерархия наследования

```
CObject  
CExpertBase  
CExpert
```

### Методы класса по группам

Инициализация	
<a href="#">Init</a>	Инициализирует экземпляр класса
<a href="#">virtual InitSignal</a>	Инициализирует объект торговых сигналов
<a href="#">virtual InitTrailing</a>	Инициализирует объект трейлинга

<code>virtual InitMoney</code>	Инициализирует объект управления капиталом
<code>virtual InitTrade</code>	Инициализирует объект торговли
<code>virtual ValidationSettings</code>	Проверяет корректность настроек
<code>virtual InitIndicators</code>	Инициализирует необходимые индикаторы и таймсерии
<code>virtual InitParameters</code>	Инициализирует параметры эксперта
<code>virtual Deinit</code>	Деинициализирует экземпляр класса
<code>virtual DeinitSignal</code>	Деинициализирует объект торговых сигналов
<code>virtual DeinitTrailing</code>	Деинициализирует объект трейлинга
<code>virtual DeinitMoney</code>	Деинициализирует объект управления капиталом
<code>virtual DeinitTrade</code>	Деинициализирует объект торговли
<code>virtual DeinitIndicators</code>	Деинициализирует индикаторы и таймсерии
<b>Установка параметров</b>	
<code>Magic</code>	Устанавливает идентификатор эксперта
<code>MaxOrders</code>	Получает/Устанавливает максимально допустимое количество ордеров
<code>OnTickProcess</code>	Устанавливает флаг обработки события "OnTick"
<code>OnTradeProcess</code>	Устанавливает флаг обработки события "OnTrade"
<code>OnTimerProcess</code>	Устанавливает флаг обработки события "OnTimer"
<code>OnChartEventProcess</code>	Устанавливает флаг обработки события "OnChartEvent"
<code>OnBookEventProcess</code>	Устанавливает флаг обработки события "OnBookEvent"
<b>Обработчики событий</b>	
<code>OnTick</code>	Обработчик события OnTick
<code>OnTrade</code>	Обработчик события OnTrade
<code>OnTimer</code>	Обработчик события OnTimer
<code>OnChartEvent</code>	Обработчик события OnChartEvent
<code>OnBookEvent</code>	Обработчик события OnBookEvent
<b>Обновление</b>	

<a href="#"><u>Refresh</u></a>	Обновляет все необходимые данные
<b>Основной метод обработки</b>	
<a href="#"><u>Processing</u></a>	Реализует основной алгоритм
<b>Методы входа в рынок</b>	
<a href="#"><u>CheckOpen</u></a>	Проверяет необходимость и возможность входа в рынок
<a href="#"><u>CheckOpenLong</u></a>	Проверяет необходимость и возможность входа в длинную позицию
<a href="#"><u>CheckOpenShort</u></a>	Проверяет необходимость и возможность входа в короткую позицию
<a href="#"><u>OpenLong</u></a>	Выполняет операции для открытия длинной позиции
<a href="#"><u>OpenShort</u></a>	Выполняет операции для открытия короткой позиции
<b>Методы выхода из рынка</b>	
<a href="#"><u>CheckClose</u></a>	Проверяет необходимость выхода из рынка
<a href="#"><u>CheckCloseLong</u></a>	Проверяет необходимость выхода из длинной позиции
<a href="#"><u>CheckCloseShort</u></a>	Проверяет необходимость выхода из короткой позиции
<a href="#"><u>CloseAll</u></a>	Выходит из рынка полностью
<a href="#"><u>Close</u></a>	Выходит из рынка
<a href="#"><u>CloseLong</u></a>	Выходит из длинной позиции
<a href="#"><u>CloseShort</u></a>	Выходит из короткой позиции
<b>Методы разворота позиции</b>	
<a href="#"><u>CheckReverse</u></a>	Проверяет необходимость разворота позиции
<a href="#"><u>CheckReverseLong</u></a>	Проверяет необходимость разворота длинной позиции
<a href="#"><u>CheckReverseShort</u></a>	Проверяет необходимость разворота короткой позиции
<a href="#"><u>ReverseLong</u></a>	Производит разворот длинной позиции
<a href="#"><u>ReverseShort</u></a>	Производит разворот короткой позиции
<b>Методы сопровождения позиций/ордеров</b>	
<a href="#"><u>CheckTrailingStop</u></a>	Проверяет необходимость модификации параметров позиции

<a href="#">CheckTrailingStopLong</a>	Проверяет необходимость модификации параметров длинной позиции
<a href="#">CheckTrailingStopShort</a>	Проверяет необходимость модификации параметров короткой позиции
<a href="#">TrailingStopLong</a>	Модифицирует параметры длинной позиции
<a href="#">TrailingStopShort</a>	Модифицирует параметры короткой позиции
<a href="#">CheckTrailingOrderLong</a>	Проверяет необходимость модификации параметров ордера на покупку
<a href="#">CheckTrailingOrderShort</a>	Проверяет необходимость модификации параметров ордера на продажу
<a href="#">TrailingOrderLong</a>	Модифицирует параметры ордера на покупку
<a href="#">TrailingOrderShort</a>	Модифицирует параметры ордера на продажу
<b>Методы удаления ордеров</b>	
<a href="#">CheckDeleteOrderLong</a>	Проверяет необходимость удаления ордеров на покупку
<a href="#">CheckDeleteOrderShort</a>	Проверяет необходимость удаления ордеров на продажу
<a href="#">DeleteOrders</a>	Удаляет все ордера
<a href="#">DeleteOrder</a>	Удаляет ордер
<a href="#">DeleteOrderLong</a>	Удаляет ордер на покупку
<a href="#">DeleteOrderShort</a>	Удаляет ордер на продажу
<b>Методы определения объема</b>	
<a href="#">LotOpenLong</a>	Определяет объем торговой операции на покупку
<a href="#">LotOpenShort</a>	Определяет объем торговой операции на продажу
<a href="#">LotReverse</a>	Определяет объем торговой операции разворота позиции
<b>Методы работы с торговой историей</b>	
<a href="#">PrepareHistoryDate</a>	Устанавливает начальную дату контролируемой истории торговли
<a href="#">HistoryPoint</a>	Создает контрольную точку истории торговли
<a href="#">CheckTradeState</a>	Обрабатывает изменение истории торговли
<b>Методы работы с флагами торговых событий</b>	

<a href="#">WaitEvent</a>	Устанавливает флаг ожидания торгового события
<a href="#">NoWaitEvent</a>	Сбрасывает флаг ожидания торгового события
<b>Методы обработки торговых событий</b>	
<a href="#">TradeEventPositionStopTake</a>	Обработчик события "Срабатывание Stop Loss/Take Profit"
<a href="#">TradeEventOrderTriggered</a>	Обработчик события "Отложенный ордер сработал"
<a href="#">TradeEventPositionOpened</a>	Обработчик события "Открытие позиции"
<a href="#">TradeEventPositionVolumeChanged</a>	Обработчик события "Добавление/уменьшение позиции"
<a href="#">TradeEventPositionModified</a>	Обработчик события "Модификация параметров позиции"
<a href="#">TradeEventPositionClosed</a>	Обработчик события "Закрытие позиции"
<a href="#">TradeEventOrderPlaced</a>	Обработчик события "Отложенный ордер установлен"
<a href="#">TradeEventOrderModified</a>	Обработчик события "Отложенный ордер модифицирован"
<a href="#">TradeEventOrderDeleted</a>	Обработчик события "Отложенный ордер удален"
<a href="#">TradeEventNotIdentified</a>	Обработчик неидентифицированного события
<b>Служебные методы</b>	
<a href="#">TimeframeAdd</a>	Добавляет таймфрейм для контроля
<a href="#">TimeframesFlags</a>	Формирует флаги таймфреймов
<a href="#">SelectPosition</a>	Выбор позиции для последующей работы с ней.

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

**Методы унаследованные от CExpertBase**

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#)

## Init

Инициализирует экземпляр класса.

```
bool Init(
    string          symbol,           // символ
    ENUM_TIMEFRAMES period,          // период
    bool            every_tick,       // флаг
    ulong           magic             // идентификатор эксперта
)
```

### Параметры

*symbol*

[in] Рабочий символ эксперта.

*period*

[in] Рабочий период эксперта из перечисления [ENUM\\_TIMEFRAMES](#).

*every\_tick*

[in] Флаг.

*magic*

[in] Иденитификатор эксперта.

### Возвращаемое значение

Нет.

### Примечание

Если параметр *every\_tick* установлен в `true`, то вызов метода [Processing\(\)](#) производится на каждом тике рабочего символа. В противном случае вызов метода [Processing\(\)](#) будет производиться только при формировании нового бара на рабочем периоде рабочего символа эксперта.

## Magic

Устанавливает идентификатор (magic) эксперта.

```
void Magic(
    ulong value      // новое значение
)
```

### Параметры

*value*

[in] Новое значение идентификатора эксперта.

### Возвращаемое значение

Нет.

### Примечание

При изменении идентификатора эксперта всем вспомогательным объектам присваивается то же значение идентификатора.

### Реализация

```
//+-----+
//| Sets magic number for object and its dependent objects
//| INPUT: value - new value of magic number.
//| OUTPUT: no.
//| REMARK: no.
//+-----+
void CExpert::Magic(ulong value)
{
    if(m_trade!=NULL)    m_trade.SetExpertMagicNumber(value);
    if(m_signal!=NULL)   m_signal.Magic(value);
    if(m_money!=NULL)    m_money.Magic(value);
    if(m_trailing!=NULL) m_trailing.Magic(value);
//--
    CExpertBase::Magic(value);
}
```

## InitSignal

Инициализирует объект торговых сигналов.

```
virtual bool InitSignal(
    CExpertSignal*     signal=NULL,           // указатель
)
```

### Параметры

*signal*

[in] Указатель на объект [CExpertSignal](#) или его потомка.

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Если параметр *signal* будет равен NULL, то объект торговых сигналов будет проинициализирован базовым классом [CExpertSignal](#) (который ничего не делает).

## InitTrailing

Инициализирует объект трейлинга.

```
virtual bool InitTrailing(
    CExpertTrailing* trailing=NULL,           // указатель
)
```

### Параметры

*trailing*

[in] Указатель на объект [CExpertTrailing](#) или его потомка.

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Если параметр *trailing* будет равен NULL, то объект трейлинга будет проинициализирован базовым классом [ExpertTrailing](#) (который ничего не делает).

## InitMoney

Инициализирует объект управления капиталом.

```
virtual bool InitMoney(
    CExpertMoney* money=NULL,           // указатель
)
```

### Параметры

*money*

[in] Указатель на объект [CExpertMoney](#) или его потомка.

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Если параметр *money* будет равен NULL, то объект управления капиталом будет проинициализирован базовым классом [CExpertMoney](#) (который всегда предлагает минимальный лот).

## InitTrade

Инициализирует объект торговли.

```
virtual bool InitTrade(  
    ulong      magic,           // идентификатор  
    CExpertTrade* trade=NULL // указатель  
)
```

### Параметры

*magic*

[in] Идентификатор эксперта, который будет использоваться при формировании торговых ордеров.

*trade*

[in] Указатель на объект торговли.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Init

Деинициализирует экземпляр класса.

```
virtual void Deinit()
```

### Возвращаемое значение

Нет.

## OnTickProcess

Устанавливает флаг обработки события [OnTick](#).

```
void OnTickOProcess(  
    bool     value        // флаг  
)
```

### Параметры

*value*

[in] Флаг обработки события [OnTick](#).

### Возвращаемое значение

Нет.

### Примечание

При установке флага в состояние true, событие [OnTick](#) будет обрабатываться, в противном случае событие [OnTick](#) обрабатываться не будет. По умолчанию флаг установлен в состояние true.

## OnTradeProcess

Устанавливает флаг обработки события [OnTrade](#).

```
void OnTradeProcess(  
    bool     value        // флаг  
)
```

### Параметры

*value*

[in] Флаг обработки события [OnTrade](#).

### Возвращаемое значение

Нет.

### Примечание

При установке флага в состояние true, событие [OnTrade](#) будет обрабатываться, в противном случае событие [OnTrade](#) обрабатываться не будет. По умолчанию флаг установлен в состояние false.

## OnTimerProcess

Устанавливает флаг обработки события [OnTimer](#).

```
void OnTimerProcess(
    bool     value        // флаг
)
```

### Параметры

*value*

[in] Флаг обработки события [OnTimer](#).

### Возвращаемое значение

Нет.

### Примечание

При установке флага в состояние true, событие [OnTimer](#) будет обрабатываться, в противном случае событие [OnTimer](#) обрабатываться не будет. По умолчанию флаг установлен в состояние false.

## OnChartEventProcess

Устанавливает флаг обработки события [OnChartEvent](#).

```
void OnChartEventProcess(
    bool     value        // флаг
)
```

### Параметры

*value*

[in] Флаг обработки события [OnChartEvent](#).

### Возвращаемое значение

Нет.

### Примечание

При установке флага в состояние true, событие [OnChartEvent](#) будет обрабатываться, в противном случае событие [OnChartEvent](#) обрабатываться не будет. По умолчанию флаг установлен в состояние false.

## OnBookEventProcess

Устанавливает флаг обработки события [OnBookEvent](#).

```
void OnChartEventProcess(
    bool     value        // флаг
)
```

### Параметры

*value*

[in] Флаг обработки события [OnBookEvent](#).

### Возвращаемое значение

Нет.

### Примечание

При установке флага в состояние true, событие [OnBookEvent](#) будет обрабатываться, в противном случае событие [OnBookEvent](#) обрабатываться не будет. По умолчанию флаг установлен в состояние false.

## MaxOrders (Get Method)

Получает максимально допустимое количество ордеров.

```
int MaxOrders()
```

### Возвращаемое значение

Максимально допустимое количество ордеров.

## MaxOrders (Set Method)

Устанавливает максимально допустимое количество ордеров.

```
void MaxOrders(  
    int      max_orders           // количество ордеров  
)
```

### Параметры

*max\_orders*

[in] Новое значение максимально допустимого количества ордеров.

### Возвращаемое значение

Нет.

### Примечание

По умолчанию максимально допустимое количество ордеров равно 1.

## Signal

Возвращает указатель на объект торговых сигналов.

```
CExpertSignal* Signal() const
```

### Возвращаемое значение

Указатель на объект торговых сигналов.

## ValidationSettings

Проверяет корректность настроек.

```
virtual bool ValidationSettings()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Проверяется корректность настроек всех вспомогательных объектов эксперта.

## InitIndicators

Инициализирует необходимые индикаторы и таймсерии.

```
virtual bool InitIndicators(
    CIndicators* indicators=NULL      // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true-в случае успешного завершения, иначе false.

### Примечание

Необходимые таймсерии инициализируются в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации.

Производится инициализация индикаторов и таймсерий всех вспомогательных объектов эксперта.

## OnTick

Обработчик события [OnTick](#).

```
virtual void OnTick()
```

**Возвращаемое значение**

Нет.

## OnTrade

Обработчик события [OnTrade](#).

```
virtual void OnTrade()
```

**Возвращаемое значение**

Нет.

## OnTimer

Обработчик события [OnTimer](#).

```
virtual void OnTimer()
```

**Возвращаемое значение**

Нет.

## OnChartEvent

Обработчик события [OnChartEvent](#).

```
virtual void OnChartEvent(
    const int      id,          // идентификатор события
    const long&   lparam,       // параметр события типа long
    const double   dparam,      // параметр события типа double
    const string   sparam       // параметр события типа string
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Параметр события типа long.

*dparam*

[in] Параметр события типа double.

*sparam*

[in] Параметр события типа string.

### Возвращаемое значение

Нет.

## OnBookEvent

Обработчик события [OnBookEvent](#).

```
virtual void OnBookEvent(
    const string& symbol           // символ
)
```

### Параметры

*symbol*

[in] Символ, по которому пришло событие [OnBookEvent](#).

### Возвращаемое значение

Нет.

## InitParameters

Инициализирует параметры эксперта.

```
virtual bool InitParameters()
```

### Возвращаемое значение

true - в случае удачи, иначе false.

### Примечание

В классе [CExpert](#) ничего не делает и всегда возвращает true.

## DeinitTrade

Деинициализирует объект торговли.

```
virtual void DeinitTrade()
```

### Возвращаемое значение

Нет.

## DeinitSignal

Деинициализирует объект объект торговых сигналов.

```
virtual void DeinitSignal()
```

### Возвращаемое значение

Нет.

## DeinitTrailing

Деинициализирует объект трейлинга.

```
virtual void DeinitTrailing()
```

### Возвращаемое значение

Нет.

## DeinitMoney

Деинициализирует объект управления капиталом.

```
virtual void DeinitMoney()
```

### Возвращаемое значение

Нет.

## DeinitIndicators

Деинициализирует индикаторы и таймсерии.

```
virtual void DeinitIndicators()
```

### Возвращаемое значение

Нет.

### Примечание

Производится деинициализация индикаторов и таймсерий всех вспомогательных объектов эксперта.

## Refresh

Обновляет все необходимые данные.

```
virtual bool Refresh()
```

### Возвращаемое значение

true, если нужна дальнейшая обработка тика, иначе - false.

### Примечание

Определяет необходимость и возможность обработки. Если обработка тика не нужна (или невозможна), возвращает false. Если обработка нужна, обновляет котировки и данные всех индикаторов и таймсерий и возвращает true.

### Реализация

```
//----------------------------------------------------------------------------------------------------------------+
//| Refreshing data for processing
//| INPUT: no.
//| OUTPUT: true-if successful, false otherwise.
//| REMARK: no.
//----------------------------------------------------------------------------------------------------------------+
bool CExpert::Refresh()
{
    MqlDateTime time;
    //--- refresh rates
    if(!m_symbol.RefreshRates()) return(false);
    //--- check need processing
    TimeToStruct(m_symbol.Time(),time);
    if(m_period_flags!=WRONG_VALUE && m_period_flags!=0)
        if((m_period_flags & TimeframesFlags(time))==0) return(false);
    m_last_tick_time=time;
    //--- refresh indicators
    m_indicators.Refresh();
    //--- ok
    return(true);
}
```

## Processing

Реализует основной алгоритм.

```
virtual bool Processing()
```

### Возвращаемое значение

true - выполнена какая-либо торговая операция, иначе - false.

### Примечание

Метод базового класса выполняет следующие действия:

1. Проверяется наличие открытой позиции по рабочему символу. Если позиции нет, то шаги 2, 3 и 4 пропускаются.
2. Проверяется необходимость переворота позиции (вызов метода [CheckReverse\(\)](#)). Если позиция "перевернута", то уходим.
3. Проверяется необходимость закрытия позиции (вызов метода [CheckClose\(\)](#)). Если позиция закрыта, то шаг 4 пропускается.
4. Проверяется необходимость модификации позиции (вызов метода [CheckTrailingStop\(\)](#)). Если позиция модифицирована, то уходим.
5. Проверяется наличие отложенных ордеров по рабочему символу. Если ордеров нет, то переходим к п.9.
6. Проверяется необходимость удалить ордер (вызов метода [CheckDeleteOrderLong\(\)](#) для ордеров на покупку или метода [CheckDeleteOrderShort\(\)](#) для ордеров на продажу). Если ордер удален, то переходим к п.9.
7. Проверяется необходимость модифицировать ордер (вызов метода [CheckTrailingOrderLong\(\)](#) для ордеров на покупку или метода [CheckTrailingOrderShort\(\)](#) для ордеров на продажу). Если ордер модифицирован, то уходим.
8. Уходим.
9. Проверяется необходимость входа в рынок (вызов метода [CheckOpen\(\)](#)).

Для реализации собственного алгоритма следует переопределить метод [Processing\(\)](#) в классе-наследнике.

### Реализация:

```
//+-----+
//| Main function
//| INPUT: no.
//| OUTPUT: true-if any trade operation processed, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::Processing()
{
//--- check if open positions
if(m_position.Select(m_symbol.Name()))
{
//--- open position is available
//--- check the possibility of reverse the position
if(CheckReverse()) return(true);
//--- check the possibility of closing the position/delete pending orders
if(!CheckClose())
{
//--- check the possibility of modifying the position
if(CheckTrailingStop()) return(true);
//--- return without operations
return(false);
}
}
//--- check if placed pending orders
int total=OrdersTotal();
if(total!=0)
{
for(int i=total-1;i>=0;i--)
{
m_order.SelectByIndex(i);
if(m_order.Symbol()!=m_symbol.Name()) continue;
if(m_order.OrderType()==ORDER_TYPE_BUY_LIMIT || m_order.OrderType()==ORDER_TY
{
//--- check the ability to delete a pending order to buy
if(CheckDeleteOrderLong()) return(true);
//--- check the possibility of modifying a pending order to buy
if(CheckTrailingOrderLong()) return(true);
}
else
{
//--- check the ability to delete a pending order to sell
if(CheckDeleteOrderShort()) return(true);
//--- check the possibility of modifying a pending order to sell
if(CheckTrailingOrderShort()) return(true);
}
}
//--- return without operations
return(false);
}
}
//--- check the possibility of opening a position/setting pending order
if(CheckOpen()) return(true);
//--- return without operations
return(false);
}
```

## SelectPosition

Выбор позиции для последующей работы с ней.

```
void SelectPosition()
```

### Возвращаемое значение

Отсутствует.

### Реализация

```
//+-----+
//| Position select
//| INPUT: no.
//| OUTPUT: no.
//| REMARK: no.
//+-----+
bool CExpert::SelectPosition(void)
{
    bool res=false;
//---
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
        res=m_position.SelectByMagic(m_symbol.Name(),m_magic);
    else
        res=m_position.Select(m_symbol.Name());
//---
    return(res);
}
```

## CheckOpen

Проверяет необходимость и возможность входа в рынок.

```
virtual bool CheckOpen()
```

### Возвращаемое значение

true - выполнена какая-либо торговая операция, иначе - false.

### Примечание

Последовательно проверяется необходимость входа в рынок в длинную позицию (вызов метода [CheckOpenLong\(\)](#)), затем в короткую (вызов метода [CheckOpenShort\(\)](#)).

### Реализация

```
//----------------------------------------------------------------------------+  
//| Check for position open or limit/stop order set  
//| INPUT: no.  
//| OUTPUT: true-if trade operation processed, false otherwise.  
//| REMARK: no.  
//----------------------------------------------------------------------------+  
bool CExpert::CheckOpen()  
{  
    if(CheckOpenLong()) return(true);  
    if(CheckOpenShort()) return(true);  
    //--- return without operations  
    return(false);  
}
```

## CheckOpenLong

Проверяет необходимость и возможность входа в рынок в длинную позицию.

```
virtual bool CheckOpenLong()
```

### Возвращаемое значение

true - выполнена какая-либо торговая операция, иначе - false.

### Примечание

Проверяется необходимость входа в рынок в длинную позицию (вызов метода `CheckOpenLong()` объекта торговых сигналов) и если условие входа выполнено, входит в рынок с параметрами, установленными объектом торговых сигналов (вызов метода [OpenLong\(\)](#)).

### Реализация

```
//+-----+
//| Check for long position open or limit/stop order set |
//| INPUT: no. |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no. |
//+-----+
bool CExpert::CheckOpenLong()
{
    double    price=EMPTY_VALUE;
    double    sl=0.0;
    double    tp=0.0;
    datetime expiration=TimeCurrent();
//--- check signal for long enter operations
    if(m_signal.CheckOpenLong(price,sl,tp,expiration))
    {
        if(!m_trade.SetOrderExpiration(expiration))
        {
            m_expiration=expiration;
        }
        return(OpenLong(price,sl,tp));
    }
//--- return without operations
    return(false);
}
```

## CheckOpenShort

Проверяет необходимость и возможность входа в рынок в короткую позицию.

```
virtual bool CheckOpenShort()
```

### Возвращаемое значение

true - выполнена какая-либо торговая операция, иначе - false.

### Примечание

Проверяется необходимость входа в рынок в короткую позицию (вызов метода `CheckOpenShort()` объекта торговых сигналов), и если условие входа выполнено, входит в рынок с параметрами, установленными объектом торговых сигналов (вызов метода [OpenShort\(\)](#)).

### Реализация

```
//+-----+
//| Check for short position open or limit/stop order set
//| INPUT: no.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::CheckOpenShort()
{
    double    price=EMPTY_VALUE;
    double    sl=0.0;
    double    tp=0.0;
    datetime  expiration=TimeCurrent();
//--- check signal for short enter operations
    if(m_signal.CheckOpenShort(price,sl,tp,expiration))
    {
        if(!m_trade.SetOrderExpiration(expiration))
        {
            m_expiration=expiration;
        }
        return(OpenShort(price,sl,tp));
    }
//--- return without operations
    return(false);
}
```

## OpenLong

Выполняет операции для входа в рынок в длинную позицию.

```
virtual bool OpenLong(
    double price,      // цена
    double sl,         // Stop Loss
    double tp          // Take Profit
)
```

### Параметры

*price*

[in] Цена входа в рынок.

*sl*

[in] Цена Stop Loss.

*tp*

[in] Цена Take Profit.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Размер лота для входа определяется вызовом метода [LotOpenLong\(\)](#). Если лот не равен 0.0, входит в рынок (вызов метода Buy(...) объекта торговли).

### Реализация

```
//+-----+
//| Long position open or limit/stop order set
//| INPUT:  price - price,
//|         sl    - stop loss,
//|         tp    - take profit.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::OpenLong(double price,double sl,double tp)
{
    if(price==EMPTY_VALUE) return(false);
    //--- get lot for open
    double lot=LotOpenLong(price,sl);
    //--- check lot for open
    if(lot==0.0) return(false);
    //---
    return(m_trade.Buy(lot,price,sl,tp));
}
```

## OpenShort

Выполняет операции для входа в рынок в короткую позицию.

```
virtual bool OpenShort(
    double price,      // цена
    double sl,         // Stop Loss
    double tp          // Take Profit
)
```

### Параметры

*price*

[in] Цена входа в рынок.

*sl*

[in] Цена Stop Loss.

*tp*

[in] Цена Take Profit.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Размер лота для входа определяется вызовом метода [LotOpenShort\(\)](#). Если лот не равен 0.0, входит в рынок (вызов метода Sell(...) объекта торговли).

### Реализация

```
//+-----+
//| Short position open or limit/stop order set
//| INPUT:  price - price,
//|         sl    - stop loss,
//|         tp    - take profit.
//| OUTPUT: true-if trade operation successful, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::OpenShort(double price,double sl,double tp)
{
    if(price==EMPTY_VALUE) return(false);
    //--- get lot for open
    double lot=LotOpenShort(price,sl);
    //--- check lot for open
    if(lot==0.0) return(false);
    //---
    return(m_trade.Sell(lot,price,sl,tp));
}
```

## CheckReverse

Проверяет необходимость и возможность разворота позиции.

```
virtual bool CheckReverse()
```

### Возвращаемое значение

true - выполнена какая-либо торговая операция, иначе - false.

### Примечание

Последовательно проверяется необходимость разворота длинной позиции (вызов метода [CheckReverseLong\(\)](#)), затем короткой (вызов метода [CheckReverseShort\(\)](#)).

### Реализация

```
//------------------------------------------------------------------------------+
//| Check for position reverse                                         |
//| INPUT: no.                                                       |
//| OUTPUT: true-if trade operation processed, false otherwise.   |
//| REMARK: no.                                                       |
//----------------------------------------------------------------------------+
bool CExpert::CheckReverse()
{
    if(m_position.PositionType() == POSITION_TYPE_BUY)
    {
        //--- check the possibility of reverse the long position
        if(CheckReverseLong()) return(true);
    }
    else
        //--- check the possibility of reverse the short position
        if(CheckReverseShort()) return(true);
    //--- return without operations
    return(false);
}
```

## CheckReverseLong

Проверяет необходимость и возможность разворота длинной позиции.

```
virtual bool CheckReverseLong()
```

### Возвращаемое значение

true - выполнена какая-либо торговая операция, иначе - false.

### Примечание

Проверяется необходимость разворота длинной позиции (вызов метода `CheckReverseLong()` объекта торговых сигналов) и если условие разворота выполнено, осуществляет разворот длинной позиции с параметрами, установленными объектом торговых сигналов (вызов метода [ReverseLong\(\)](#)).

### Реализация

```
//----------------------------------------------------------------------------------------------------------------+
//| Check for long position reverse
//| INPUT: no.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//----------------------------------------------------------------------------------------------------------------+
bool CExpert::CheckReverseLong()
{
    double    price=EMPTY_VALUE;
    double    sl=0.0;
    double    tp=0.0;
    datetime expiration=TimeCurrent();
    //--- check signal for long reverse operations
    if(m_signal.CheckReverseLong(price,sl,tp,expiration)) return(ReverseLong(price,sl,ti
    //--- return without operations
    return(false);
}
```

## CheckReverseShort

Проверяет необходимость и возможность разворота короткой позиции.

```
virtual bool CheckReverseLong()
```

### Возвращаемое значение

true - выполнена какая-либо торговая операция, иначе - false.

### Примечание

Проверяется необходимость разворота короткой позиции (вызов метода `CheckReverseShort()` объекта торговых сигналов) и если условие разворота выполнено, осуществляет разворот короткой позиции с параметрами, установленными объектом торговых сигналов (вызов метода [ReverseShort\(\)](#)).

### Реализация

```
//----------------------------------------------------------------------------------------------------------------+
//| Check for short position reverse
//| INPUT: no.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//----------------------------------------------------------------------------------------------------------------+
bool CExpert::CheckReverseShort()
{
    double    price=EMPTY_VALUE;
    double    sl=0.0;
    double    tp=0.0;
    datetime expiration=TimeCurrent();
    //--- check signal for short reverse operations
    if(m_signal.CheckReverseShort(price,sl,tp,expiration)) return(ReverseShort(price,sl,tp,expiration));
    //--- return without operations
    return(false);
}
```

## ReverseLong

Выполняет операции разворота длинной позиции.

```
virtual bool ReverseLong(
    double    price,      // цена
    double    sl,         // Stop Loss
    double    tp          // Take Profit
)
```

### Параметры

*price*

[in] Цена входа в рынок.

*sl*

[in] Цена Stop Loss.

*tp*

[in] Цена Take Profit.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Размер лота для разворота длинной позиции определяется вызовом метода [LotReverse\(\)](#). Если лот не равен 0.0, производит разворот длинной позиции (вызов метода Sell(...) объекта торговли).

В "хеджинговом" режиме учета позиций разворот осуществляется закрытием существующей позиции и открытием новой позиции противоположного направления с остаточным объемом.

### Реализация

```
//+-----+
//| Long position reverse
//| INPUT:  price - price,
//|         sl   - stop loss,
//|         tp   - take profit.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::ReverseLong(double price,double sl,double tp)
{
    if(price==EMPTY_VALUE)
        return(false);
    //--- get lot for reverse
    double lot=LotReverse(sl);
    //--- check lot
    if(lot==0.0)
        return(false);
    //---
    bool result=true;
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
    {
        //--- first close existing position
        lot-=m_position.Volume();
        result=m_trade.PositionCloseByTicket(m_position.Identifier());
    }
    if(result)
        result=m_trade.Sell(lot,price,sl,tp);
    //---
    return(result);
}
```

## ReverseShort

Выполняет операции разворота короткой позиции.

```
virtual bool ReverseShort(
    double    price,      // цена
    double    sl,         // Stop Loss
    double    tp          // Take Profit
)
```

### Параметры

*price*

[in] Цена входа в рынок.

*sl*

[in] Цена Stop Loss.

*tp*

[in] Цена Take Profit.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Размер лота для разворота короткой позиции определяется вызовом метода [LotReverse\(\)](#). Если лот не равен 0.0, производит разворот короткой позиции (вызов метода [Buy\(...\)](#) объекта торговли).

В "хеджинговом" режиме учета позиций разворот осуществляется закрытием существующей позиции и открытием новой позиции противоположного направления с остаточным объемом.

### Реализация

```
//+-----+
//| Short position reverse
//| INPUT:  price - price,
//|         sl   - stop loss,
//|         tp   - take profit.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::ReverseShort(double price,double sl,double tp)
{
    if(price==EMPTY_VALUE)
        return(false);
//--- get lot for reverse
    double lot=LotReverse(sl);
//--- check lot
    if(lot==0.0)
        return(false);
//---
    bool result=true;
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
    {
//--- first close existing position
        lot-=m_position.Volume();
        result=m_trade.PositionCloseByTicket(m_position.Identifier());
    }
    if(result)
        result=m_trade.Buy(lot,price,sl,tp);
//---
    return(result);
}
```

## CheckClose

Проверяет необходимость выхода из рынка.

```
virtual bool CheckClose()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

1. Проверяет необходимость выхода из рынка по программному Stop Out (вызов метода [CheckClose\(\)](#) объекта управления капиталом). Если условие выполняется, закрываем позицию, удаляем все ордера (вызов метода [CloseAll\(\)](#) и уходим).
2. Проверяет необходимость закрытия длинной или короткой позиции (вызов метода [CheckCloseLong\(\)](#) или [CheckCloseShort\(\)](#) соответственно) и, если позиция закрыта, удаляет все ордера (вызов метода [DeleteOrders\(\)](#)).

### Реализация

```
//---------------------------------------------------------------------+
//| Check for position close or limit/stop order delete           |
//| INPUT: no.                                                       |
//| OUTPUT: true-if trade operation processed, false otherwise.   |
//| REMARK: no.                                                      |
//---------------------------------------------------------------------+
bool CExpert::CheckClose()
{
    double lot;
    //--- position must be selected before call
    if((lot=m_money.CheckClose(GetPointer(m_position))) !=0.0)
        return(CloseAll(lot));
    //--- check for position type
    if(m_position.PositionType() == POSITION_TYPE_BUY)
    {
        //--- check the possibility of closing the long position / delete pending orders
        if(CheckCloseLong())
        {
            DeleteOrders();
            return(true);
        }
    }
    else
    {
        //--- check the possibility of closing the short position / delete pending orders
        if(CheckCloseShort())
        {
            DeleteOrders();
            return(true);
        }
    }
    //--- return without operations
    return(false);
}
```

## CheckCloseLong

Проверяет необходимость выхода из длинной позиции.

```
virtual bool CheckCloseLong()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Проверяет необходимость закрытия длинной позиции (вызов метода `CheckCloseLong()` объекта торговых сигналов) и, если условие выполнено, закрывает позицию (вызов метода [CloseLong\(\)](#)).

### Реализация

```
//----------------------------------------------------------------------------+  
//| Check for long position close or limit/stop order delete           |  
//| INPUT: no.                                                       |  
//| OUTPUT: true-if trade operation processed, false otherwise.        |  
//| REMARK: no.                                                       |  
//----------------------------------------------------------------------------+  
bool CExpert::CheckCloseLong()  
{  
    double price=EMPTY_VALUE;  
    //--- check for long close operations  
    if(m_signal.CheckCloseLong(price))  
        return(CloseLong(price));  
    //--- return without operations  
    return(false);  
}
```

## CheckCloseShort

Проверяет необходимость выхода из короткой позиции.

```
virtual bool CheckCloseShort()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Проверяет необходимость закрытия короткой позиции (вызов метода `CheckCloseShort()` объекта торговых сигналов) и, если условие выполнено, закрывает позицию (вызов метода [CloseShort\(\)](#)).

### Реализация

```
//----------------------------------------------------------------------------+  
//| Check for short position close or limit/stop order delete           |  
//| INPUT: no.                                                       |  
//| OUTPUT: true-if trade operation processed, false otherwise.          |  
//| REMARK: no.                                                       |  
//----------------------------------------------------------------------------+  
bool CExpert::CheckCloseShort()  
{  
    double price=EMPTY_VALUE;  
    //--- check for short close operations  
    if(m_signal.CheckCloseShort(price))  
        return(CloseShort(price));  
    //--- return without operations  
    return(false);  
}
```

## CloseAll

Выходит из рынка полностью или частично.

```
virtual bool CloseAll(
    double     lot      // лот
)
```

### Параметры

*lot*

[in] Размер лота, на который нужно сократить позицию.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

В "неттинговом" режиме учета позиций закрытие производится методами CExpertTrade::Buy или CExpertTrade::Sell. В "хеджинговом" режиме - методом CTrade::PositionClose, который также можно использовать и на счетах с неттинговой системой учета. Для удаления всех отложенных ордеров предназначен метод [DeleteOrders\(\)](#).

### Реализация

```
//+-----+
//| Position close and orders delete
//| INPUT:  lot - volume for close.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::CloseAll(double lot)
{
    bool result=false;
//--- check for close operations
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
        result=m_trade.PositionCloseByTicket(m_position.Identifier());
    else
    {
        if(m_position.PositionType()==POSITION_TYPE_BUY)
            result=m_trade.Sell(lot,0,0,0);
        else
            result=m_trade.Buy(lot,0,0,0);
    }
    result|=DeleteOrders();
//---
    return(result);
}
```

## Close

Выходит из рынка.

```
virtual bool Close()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Закрывает позицию (вызов метода торгового объекта PositionClose()).

### Реализация

```
////////////////////////////////////////////////////////////////////////+
//| Position close
//| INPUT: no.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
////////////////////////////////////////////////////////////////////////+
bool CExpert::Close()
{
    return(m_trade.PositionClose(m_symbol.Name()));
}
```

## CloseLong

Выходит из длинной позиции.

```
virtual bool CloseLong(
    double     price      // цена
)
```

### Параметры

*price*

[in] Цена входа в рынок.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

В "неттинговом" режиме учета позиций закрытие производится методами CExpertTrade::Buy или CExpertTrade::Sell. В "хеджинговом" режиме - методом CTrade::PositionCloseByTicket.

### Реализация

```
//+-----+
//| Long position close
//| INPUT: price - price for close.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::CloseLong(double price)
{
    bool result=false;
//---
    if(price==EMPTY_VALUE)
        return(false);
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
        result=m_trade.PositionCloseByTicket(m_position.Identifier());
    else
        result=m_trade.Sell(m_position.Volume(),price,0,0);
//---
    return(result);
}
```

## CloseShort

Выходит из короткой позиции.

```
virtual bool CloseShort(
    double     price      // цена
)
```

### Параметры

*price*

[in] Цена входа в рынок.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

В "неттинговом" режиме учета позиций закрытие производится методами CExpertTrade::Buy или CExpertTrade::Sell. В "хеджинговом" режиме - методом CTrade::PositionCloseByTicket.

### Реализация

```
//+-----+
//| Short position close
//| INPUT: price - price for close.
//| OUTPUT: true-if trade operation successful, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::CloseShort(double price)
{
    bool result=false;
//---
    if(price==EMPTY_VALUE)
        return(false);
    if(m_margin_mode==ACCOUNT_MARGIN_MODE_RETAIL_HEDGING)
        result=m_trade.PositionCloseByTicket(m_position.Identifier());
    else
        result=m_trade.Buy(m_position.Volume(),price,0,0);
//---
    return(result);
}
```

## CheckTrailingStop

Проверяет необходимость модификации параметров позиции.

```
virtual bool CheckTrailingStop()
```

### Возвращаемое значение

true - выполнена какая-либо торговая операция, иначе - false.

### Примечание

Проверяется необходимость модификации позиции (вызов метода [CheckTrailingStopLong\(\)](#) для длинной позиции или [CheckTrailingStopShort\(\)](#) для короткой позиции).

### Реализация

```
//----------------------------------------------------------------------------+  
//| Check for trailing stop/profit position  
//| INPUT: no.  
//| OUTPUT: true-if trade operation processed, false otherwise.  
//| REMARK: no.  
//----------------------------------------------------------------------------+  
bool CExpert::CheckTrailingStop()  
{  
    //--- position must be selected before call  
    if(m_position.PositionType()==POSITION_TYPE_BUY)  
    {  
        //--- check the possibility of modifying the long position  
        if(CheckTrailingStopLong()) return(true);  
    }  
    else  
    {  
        //--- check the possibility of modifying the short position  
        if(CheckTrailingStopShort()) return(true);  
    }  
    //--- return without operations  
    return(false);  
}
```

## CheckTrailingStopLong

Проверяет необходимость модификации параметров длинной позиции.

```
virtual bool CheckTrailingStopLong()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Проверяется необходимость модификации длинной позиции (вызов метода [CheckTrailingStopLong\(\)](#) объекта трейлинга). В случае выполнения условия модифицирует параметры длинной позиции в соответствии с установками объекта трейлинга (вызов метода [TrailingStopLong\(\)](#)).

### Реализация

```
//---------------------------------------------------------------------+
//| Check for trailing stop/profit long position
//| INPUT: no.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//---------------------------------------------------------------------+
bool CExpert::CheckTrailingStopLong()
{
    double sl=EMPTY_VALUE;
    double tp=EMPTY_VALUE;
    //--- check for long trailing stop operations
    if(m_trailing.CheckTrailingStopLong(GetPointer(m_position),sl,tp))
    {
        if(sl==EMPTY_VALUE) sl=m_position.StopLoss();
        if(tp==EMPTY_VALUE) tp=m_position.TakeProfit();
        //--- long trailing stop operations
        return(TrailingStopLong(sl,tp));
    }
    //--- return without operations
    return(false);
}
```

## CheckTrailingStopShort

Проверяет необходимость модификации параметров короткой позиции.

```
virtual bool CheckTrailingStopShort()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Проверяется необходимость модификации короткой позиции (вызов метода [CheckTrailingStopShort\(\)](#) объекта трейлинга). В случае выполнения условия модифицирует параметры короткой позиции в соответствии с установками объекта трейлинга (вызов метода [TrailingStopShort\(\)](#)).

### Реализация

```
//---------------------------------------------------------------------+
//| Check for trailing stop/profit short position               |
//| INPUT: no.                                                 |
//| OUTPUT: true-if trade operation processed, false otherwise. |
//| REMARK: no.                                                |
//---------------------------------------------------------------------+
bool CExpert::CheckTrailingStopShort()
{
    double sl=EMPTY_VALUE;
    double tp=EMPTY_VALUE;
    //--- check for short trailing stop operations
    if(m_trailing.CheckTrailingStopShort(GetPointer(m_position),sl,tp))
    {
        if(sl==EMPTY_VALUE) sl=m_position.StopLoss();
        if(tp==EMPTY_VALUE) tp=m_position.TakeProfit();
        //--- short trailing stop operations
        return(TrailingStopShort(sl,tp));
    }
    //--- return without operations
    return(false);
}
```

## TrailingStopLong

Модифицирует параметры длинной позиции.

```
virtual bool TrailingStopLong(
    double sl, // цена Stop Loss
    double tp, // цена Take Profit
)
```

### Параметры

*sl*

[in] Цена Stop Loss.

*tp*

[in] Цена Take Profit.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Модифицирует позицию (вызов метода PositionModify() объекта торговли).

### Реализация

```
//+-----+
//| Trailing stop/profit long position
//| INPUT: sl - new stop loss,
//|        tp - new take profit.
//| OUTPUT: true-if trade operation successful, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::TrailingStopLong(double sl,double tp)
{
    return(m_trade.PositionModify(m_symbol.Name(),sl,tp));
}
```

## TrailingStopShort

Модифицирует параметры короткой позиции.

```
virtual bool TrailingStopLong(
    double sl, // цена Stop Loss
    double tp, // цена Take Profit
)
```

### Параметры

*sl*

[in] Цена Stop Loss.

*tp*

[in] Цена Take Profit.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Модифицирует позицию (вызов метода PositionModify() объекта торговли).

### Реализация

```
//+-----+
//| Trailing stop/profit short position
//| INPUT: sl - new stop loss,
//|        tp - new take profit.
//| OUTPUT: true-if trade operation successful, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::TrailingStopShort(double sl,double tp)
{
    return(m_trade.PositionModify(m_symbol.Name(),sl,tp));
}
```

## CheckTrailingOrderLong

Проверяет необходимость модификации параметров ордера на покупку.

```
virtual bool CheckTrailingOrderLong()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Проверяется необходимость модификации параметров ордера на покупку (вызов метода `CheckTrailingOrderLong()` объекта торговых сигналов). В случае выполнения условия модифицирует параметры ордера (вызов метода [TrailingOrderLong\(\)](#)).

### Реализация

```
//----------------------------------------------------------------------------------------------------------------+
//| Check for trailing long limit/stop order
//| INPUT: no.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//----------------------------------------------------------------------------------------------------------------+
bool CExpert::CheckTrailingOrderLong()
{
    double price;
//--- check the possibility of modifying the long order
    if(m_signal.CheckTrailingOrderLong(GetPointer(m_order),price))
        return(TrailingOrderLong(m_order.PriceOpen()-price));
//--- return without operations
    return(false);
}
```

## CheckTrailingOrderShort

Проверяет необходимость модификации параметров ордера на продажу.

```
virtual bool CheckTrailingOrderShort()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Проверяется необходимость модификации параметров ордера на продажу (вызов метода `CheckTrailingOrderShort()` объекта торговых сигналов). В случае выполнения условия модифицирует параметры ордера (вызов метода [TrailingOrderShort\(\)](#)).

### Реализация

```
//----------------------------------------------------------------------------------------------------------------+
//| Check for trailing short limit/stop order
//| INPUT: no.
//| OUTPUT: true-if trade operation processed, false otherwise.
//| REMARK: no.
//----------------------------------------------------------------------------------------------------------------+
bool CExpert::CheckTrailingOrderShort()
{
    double price;
    //--- check the possibility of modifying the short order
    if(m_signal.CheckTrailingOrderShort(GetPointer(m_order),price))
        return(TrailingOrderShort(m_order.PriceOpen()-price));
    //--- return without operations
    return(false);
}
```

## TrailingOrderLong

Модифицирует параметры ордера на покупку.

```
virtual bool TrailingOrderLong(
    double delta // смещение
)
```

### Параметры

*delta*

[in] Изменение цены.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Модифицирует параметры ордера (вызов метода OrderModify(...) объекта торговли).

### Реализация

```
//+-----+
//| Trailing long limit/stop order
//| INPUT: delta - price change.
//| OUTPUT: true-if trade operation successful, false otherwise.
//| REMARK: no.
//+-----+
bool CExpert::TrailingOrderLong(double delta)
{
    ulong ticket=m_order.Ticket();
    double price =m_order.PriceOpen()-delta;
    double sl    =m_order.StopLoss()-delta;
    double tp    =m_order.TakeProfit()-delta;
    //--- modifying the long order
    return(m_trade.OrderModify(ticket,price,sl,tp,m_order.TypeTime(),m_order.TimeExpira
}
```

## TrailingOrderShort

Модифицирует параметры ордера на продажу.

```
virtual bool TrailingOrderShort(
    double delta // смещение
)
```

### Параметры

*delta*

[in] Изменение цены.

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Модифицирует параметры ордера (вызов метода OrderModify(...) объекта торговли).

### Реализация

```
//----------------------------------------------------------------------------------------------------------------+
//| Trailing short limit/stop order
//| INPUT: delta - price change.
//| OUTPUT: true-if trade operation successful, false otherwise.
//| REMARK: no.
//----------------------------------------------------------------------------------------------------------------+
bool CExpert::TrailingOrderShort(double delta)
{
    ulong ticket=m_order.Ticket();
    double price =m_order.PriceOpen()-delta;
    double sl    =m_order.StopLoss()-delta;
    double tp    =m_order.TakeProfit()-delta;
    //--- modifying the short order
    return(m_trade.OrderModify(ticket,price,sl,tp,m_order.TypeTime(),m_order.TimeExpira
}
```

## CheckDeleteOrderLong

Проверяет необходимость удаления ордера на покупку.

```
virtual bool CheckDeleteOrderLong()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

1. Проверяет время истечения ордера.
2. Проверяет необходимость удаления ордера (вызов метода `CheckCloseLong()` объекта торговых сигналов). В случае выполнения одного из условий, удаляет ордер (вызов метода [DeleteOrderLong\(\)](#)).

### Реализация

```
//---------------------------------------------------------------------+
//| Check for delete long limit/stop order                         |
//| INPUT: no.                                                       |
//| OUTPUT: true-if trade operation processed, false otherwise.   |
//| REMARK: no.                                                      |
//---------------------------------------------------------------------+
bool CExpert::CheckDeleteOrderLong()
{
    double price;
//--- check the possibility of deleting the long order
    if(m_expiration!=0 && TimeCurrent()>m_expiration)
    {
        m_expiration=0;
        return(DeleteOrderLong());
    }
    if(m_signal.CheckCloseLong(price))
        return(DeleteOrderLong());
//--- return without operations
    return(false);
}
```

## CheckDeleteOrderShort

Проверяет необходимость удаления ордера на продажу.

```
virtual bool CheckDeleteOrderShort()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

1. Проверяет время истечения ордера.
2. Проверяет необходимость удаления ордера (вызов метода `CheckCloseShort()` объекта торговых сигналов). В случае выполнения одного из условий, удаляет ордер (вызов метода [DeleteOrderShort\(\)](#)).

### Реализация

```
//---------------------------------------------------------------------+
//| Check for delete short limit/stop order                         |
//| INPUT: no.                                                       |
//| OUTPUT: true-if trade operation processed, false otherwise.    |
//| REMARK: no.                                                      |
//---------------------------------------------------------------------+
bool CExpert::CheckDeleteOrderShort()
{
    double price;
//--- check the possibility of deleting the short order
    if(m_expiration!=0 && TimeCurrent()>m_expiration)
    {
        m_expiration=0;
        return(DeleteOrderShort());
    }
    if(m_signal.CheckCloseShort(price))
        return(DeleteOrderShort());
//--- return without operations
    return(false);
}
```

## DeleteOrders

Удаляет все ордера.

```
virtual bool DeleteOrders()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Удаляет все ордера (вызов в цикле метода [DeleteOrder](#)).

### Реализация

```
////////////////////////////////////////////////////////////////////////+
//| Delete all limit/stop orders                                |
//| INPUT: no.                                                 |
//| OUTPUT: true-if trade operation successful, false otherwise. |
//| REMARK: no.                                              |
////////////////////////////////////////////////////////////////////////+
bool CExpert::DeleteOrders()
{
    bool result=false;
    int total=OrdersTotal();
//---
    for(int i=total-1;i>=0;i--)
    {
        if(m_order.Select(OrderGetTicket(i)))
        {
            if(m_order.Symbol()!=m_symbol.Name()) continue;
            result|=DeleteOrder();
        }
    }
//---
    return(result);
}
```

## DeleteOrder

Удаляет ордер.

```
virtual bool DeleteOrder()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Удаляет ордер (вызов метода OrderDelete(...) объекта торговли).

### Реализация

```
////////////////////////////////////////////////////////////////////////+
//| Delete limit/stop order
//| INPUT: no.
//| OUTPUT: true-if trade operation successful, false otherwise.
//| REMARK: no.
////////////////////////////////////////////////////////////////////////+
bool CExpert::DeleteOrder()
{
    return(m_trade.OrderDelete(m_order.Ticket()));
}
```

## DeleteOrderLong

Удаляет ордер на покупку.

```
virtual bool DeleteOrderLong()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Удаляет ордер на покупку (вызов метода OrderDelete(..) объекта торговли).

### Реализация

```
////////////////////////////////////////////////////////////////////////+
//| Delete long limit/stop order
//| INPUT: no.
//| OUTPUT: true-if trade operation successful, false otherwise.
//| REMARK: no.
////////////////////////////////////////////////////////////////////////+
bool CExpert::DeleteOrderLong()
{
    return(m_trade.OrderDelete(m_order.Ticket()));
}
```

## DeleteOrderShort

Удаляет ордер на продажу.

```
virtual bool DeleteOrderShort()
```

### Возвращаемое значение

true - выполнена торговая операция, иначе - false.

### Примечание

Удаляет ордер на продажу (вызов метода OrderDelete(...) объекта торговли).

### Реализация

```
////////////////////////////////////////////////////////////////////////+
//| Delete short limit/stop order
//| INPUT: no.
//| OUTPUT: true-if trade operation successful, false otherwise.
//| REMARK: no.
////////////////////////////////////////////////////////////////////////+
bool CExpert::DeleteOrderShort()
{
    return(m_trade.OrderDelete(m_order.Ticket()));
}
```

## LotOpenLong

Определяет объем торговой операции на покупку.

```
double LotOpenLong(
    double price,      // цена
    double sl         // цена Stop Loss
)
```

### Параметры

*price*

[in] Цена входа в рынок.

*sl*

[in] Цена Stop Loss.

### Возвращаемое значение

Объем (в лотах) для операции на покупку.

### Примечание

Определяет объем торговой операции на покупку (вызов метода CheckOpenLong(...) объекта управления капиталом).

### Реализация

```
//----------------------------------------------------------------------------------------------------------------+
//| Method of getting the lot for open long position.
//| INPUT:  price - price,
//|         sl   - stop loss.
//| OUTPUT: lot for open.
//| REMARK: no.
//----------------------------------------------------------------------------------------------------------------+
double CExpert::LotOpenLong(double price,double sl)
{
    return(m_money.CheckOpenLong(price,sl));
}
```

## LotOpenShort

Определяет объем торговой операции на продажу.

```
double LotOpenShort(
    double price,      // цена
    double sl         // цена Stop Loss
)
```

### Параметры

*price*

[in] Цена входа в рынок.

*sl*

[in] Цена Stop Loss.

### Возвращаемое значение

Объем (в лотах) для операции на продажу.

### Примечание

Определяет объем торговой операции на продажу (вызов метода CheckOpenShort(...) объекта управления капиталом).

### Реализация

```
//-------------------------------------------------------------------------------------------------
//| Method of getting the lot for open short position.
//| INPUT:  price - price,
//|        sl   - stop loss.
//| OUTPUT: lot for open.
//| REMARK: no.
//-------------------------------------------------------------------------------------------------
double CExpert::LotOpenShort(double price,double sl)
{
    return(m_money.CheckOpenShort(price,sl));
}
```

## LotReverse

Определяет объем торговой операции разворота позиции.

```
double LotReverse(
    double     sl          // цена Stop Loss
)
```

### Параметры

*sl*

[in] Цена Stop Loss.

### Возвращаемое значение

Объем (в лотах) для операции разворота позиции.

### Примечание

Определяет объем торговой операции разворота позиции (вызов метода CheckReverse(...) объекта управления капиталом).

### Реализация

```
//---------------------------------------------------------------------
//| Method of getting the lot for reverse position.
//| INPUT: sl - stop loss.
//| OUTPUT: lot for open.
//| REMARK: no.
//---------------------------------------------------------------------
double CExpert::LotReverse(double sl)
{
    return(m_money.CheckReverse(GetPointer(m_position),sl));
}
```

## PrepareHistoryDate

Устанавливает начальную дату контролируемой истории торговли.

```
void PrepareHistoryDate()
```

### Примечание

По умолчанию дата начала контролируемой истории устанавливается с начала месяца (но не менее одних суток).

## HistoryPoint

Создает контрольную точку истории торговли.

```
void HistoryPoint(
    bool from_check_trade=false // флаг
)
```

### Параметры

*from\_check\_trade=false*

[in] Флаг для предотвращения рекурсии.

### Примечание

Запоминает количество позиций, ордеров, сделок и исторических ордеров.

## CheckTradeState

Обрабатывает изменение истории торговли.

```
bool CheckTradeState()
```

### Возвращаемое значение

true - если событие обработанно, иначе - false.

### Примечание

Проверяет количество позиций, ордеров, сделок и исторических ордеров, сравнивая их с ранее сохраненными в методе [HistoryPoint\(\)](#). В случае изменения вызывает соответствующий виртуальный обработчик.

## WaitEvent

Устанавливает флаг ожидания торгового события.

```
void WaitEvent(
    ENUM_TRADE_EVENTS event           // флаг
)
```

### Параметры

*event*

[in] Флаг ожидания торгового события (из перечисления ENUM\_TRADE\_EVENTS), который необходимо установить.

### Возвращаемое значение

Нет.

### Флаги событий

```
//--- flags of expected events
enum ENUM_TRADE_EVENTS
{
    TRADE_EVENT_NO_EVENT          =0,           // no expected events
    TRADE_EVENT_POSITION_OPEN      =0x1,         // flag of expecting the "opening of
    TRADE_EVENT_POSITION_VOLUME_CHANGE=0x2,     // flag of expecting of the "modifica
    TRADE_EVENT_POSITION MODIFY   =0x4,         // flag of expecting of the "modifica
    TRADE_EVENT_POSITION_CLOSE    =0x8,         // flag of expecting of the "closing
    TRADE_EVENT_POSITION_STOP_TAKE=0x10,        // flag of expecting of the "trigger
    TRADE_EVENT_ORDER_PLACE       =0x20,        // flag of expecting of the "placing
    TRADE_EVENT_ORDER MODIFY     =0x40,        // flag of expecting of the "modifica
    TRADE_EVENT_ORDER_DELETE     =0x80,        // flag of expecting of the "deletio
    TRADE_EVENT_ORDER_TRIGGER    =0x100,       // flag of expecting of the "trigger
};
```

## NoWaitEvent

Сбрасывает флаг ожидания торгового события.

```
void NoWaitEvent(
    ENUM_TRADE_EVENTS event           // флаг
)
```

### Параметры

*event*

[in] Флаг ожидания торгового события (из перечисления ENUM\_TRADE\_EVENTS), который нужно сбросить.

### Возвращаемое значение

Нет.

### Флаги событий

```
---- flags of expected events
enum ENUM_TRADE_EVENTS
{
    TRADE_EVENT_NO_EVENT          =0,           // no expected events
    TRADE_EVENT_POSITION_OPEN      =0x1,         // flag of expecting the "opening of
    TRADE_EVENT_POSITION_VOLUME_CHANGE=0x2,   // flag of expecting of the "modifica
    TRADE_EVENT_POSITION MODIFY   =0x4,         // flag of expecting of the "modifica
    TRADE_EVENT_POSITION_CLOSE    =0x8,         // flag of expecting of the "closing
    TRADE_EVENT_POSITION_STOP_TAKE=0x10,        // flag of expecting of the "trigger
    TRADE_EVENT_ORDER_PLACE       =0x20,        // flag of expecting of the "placing
    TRADE_EVENT_ORDER MODIFY     =0x40,        // flag of expecting of the "modifica
    TRADE_EVENT_ORDER_DELETE     =0x80,        // flag of expecting of the "deletio
    TRADE_EVENT_ORDER_TRIGGER    =0x100,        // flag of expecting of the "trigger
};
```

## TradeEventPositionStopTake

Обработчик события "Срабатывание Stop Loss/Take Profit".

```
virtual bool TradeEventPositionStopTake()
```

### Возвращаемое значение

Метод класса [CExpert](#) ничего не делает и всегда возвращает true.

## TradeEventOrderTriggered

Обработчик события "Отложенный ордер сработал".

```
virtual bool TradeEventOrderTriggered()
```

### Возвращаемое значение

Метод класса [CExpert](#) ничего не делает и всегда возвращает true.

## TradeEventPositionOpened

Обработчик события "Открытие позиции".

```
virtual bool TradeEventPositionOpened()
```

### Возвращаемое значение

Метод класса [CExpert](#) ничего не делает и всегда возвращает true.

## TradeEventPositionVolumeChanged

Обработчик события "Добавление/уменьшение позиции".

```
virtual bool TradeEventPositionVolumeChanged()
```

### Возвращаемое значение

Метод класса [CExpert](#) ничего не делает и всегда возвращает true.

## TradeEventPositionModified

Обработчик события "Модификация параметров позиции".

```
virtual bool TradeEventPositionModified()
```

### Возвращаемое значение

Метод класса [CExpert](#) ничего не делает и всегда возвращает true.

## TradeEventPositionClosed

Обработчик события "Закрытие позиции".

```
virtual bool TradeEventPositionClosed()
```

### Возвращаемое значение

Метод класса [CExpert](#) ничего не делает и всегда возвращает true.

## TradeEventOrderPlaced

Обработчик события "Отложенный ордер установлен".

```
virtual bool TradeEventOrderPlaced()
```

### Возвращаемое значение

Метод класса [CExpert](#) ничего не делает и всегда возвращает true.

## TradeEventOrderModified

Обработчик события "Отложенный ордер модифицирован".

```
virtual bool TradeEventOrderModified()
```

### Возвращаемое значение

Метод класса [CExpert](#) ничего не делает и всегда возвращает true.

## TradeEventOrderDeleted

Обработчик события "Отложенный ордер удален".

```
virtual bool TradeEventOrderDeleted()
```

### Возвращаемое значение

Метод класса [CExpert](#) ничего не делает и всегда возвращает true.

## TradeEventNotIdentified

Обработчик неидентифицированного события.

```
virtual bool TradeEventNotIdentified()
```

### Возвращаемое значение

Метод класса [CExpert](#) ничего не делает и всегда возвращает true.

### Примечание

Следует отметить, что торговые события далеко не всегда бывают одиночными, чаще всего события от сервера поступают "пачками". В этом случае однозначная идентификация возникших торговых событий затруднена.

## TimeframeAdd

Добавляет таймфрейм для контроля.

```
void TimeframeAdd(
    ENUM_TIMEFRAMES period           // период
)
```

### Параметры

*period*

[in] Период (из перечисления [ENUM\\_TIMEFRAMES](#)), который необходимо контролировать.

### Возвращаемое значение

Нет.

## TimeframesFlags

Формирует флаги таймфреймов.

```
int TimeframesFlags(
    MqlDateTime& time           // ссылка
)
```

### Параметры

*time*

[in] Ссылка на структуру типа [MqlDateTime](#), в которой содержится новое время.

### Возвращаемое значение

Флаги таймфреймов, для которых пришел "новый бар".

## Класс CExpertSignal

Класс CExpertSignal является базовым классом для создания генераторов торговых сигналов, поэтому он, предоставляя интерфейсы, сам ничего не делает (исключение составляют методы [CheckReverseLong\(\)](#) и [CheckReverseShort\(\)](#)).

Для того чтобы генератор торговых сигналов "загенерировал", нужно:

1. Определиться с алгоритмами генерации торговых сигналов;
2. Создать свой класс генератора, унаследовав его от CExpertSignal;
3. Переопределить в своем классе виртуальные методы базового, заложив в них соответствующие алгоритмы.

В качестве примера можно рассмотреть любой mqh-файл из папки Expert\Signal\.

### Описание

Класс CExpertSignal является основой для реализации алгоритмов генерации торговых сигналов.

### Декларация

```
class CExpertSignal : public CExpertBase
```

### Заголовок

```
#include <Expert\ExpertSignal.mqh>
```

### Иерархия наследования

```
CObject  
CExpertBase  
CExpertSignal
```

### Прямые потомки

CSignalAC, CSignalAMA, CSignalAO, CSignalBearsPower, CSignalBullsPower, CSignalCCI, CSignalDeM, CSignalDEMA, CSignalEnvelopes, CSignalFrAMA, CSignalRSI, CSignalRVI, CSignalSAR, CSignalStoch, CSignalTEMA, CSignalTriX, CSignalWPR

### Методы класса по группам

Инициализация	
virtual <a href="#">InitIndicators</a>	Инициализирует необходимые индикаторы и таймсерии
virtual <a href="#">ValidationSettings</a>	Проверяет корректность настроек объекта
virtual <a href="#">AddFilter</a>	Добавляет фильтр в комбинированный сигнал
<b>Доступ к защищенным данным</b>	

<a href="#">BasePrice</a>	Устанавливает базовый уровень цены
<a href="#">UsedSeries</a>	Получает флаги используемых таймсерий
<b>Установка параметров</b>	
<a href="#">Weight</a>	Устанавливает значение параметра "Weight"
<a href="#">PatternsUsage</a>	Устанавливает значение параметра "PatternsUsage"
<a href="#">General</a>	Устанавливает значение параметра "General"
<a href="#">Ignore</a>	Устанавливает значение параметра "Ignore"
<a href="#">Invert</a>	Устанавливает значение параметра "Invert"
<a href="#">ThresholdOpen</a>	Устанавливает значение параметра "ThresholdOpen"
<a href="#">ThresholdClose</a>	Устанавливает значение параметра "ThresholdClose"
<a href="#">PriceLevel</a>	Устанавливает значение параметра "PriceLevel"
<a href="#">StopLevel</a>	Устанавливает значение параметра "StopLevel"
<a href="#">TakeLevel</a>	Устанавливает значение параметра "TakeLevel"
<a href="#">Expiration</a>	Устанавливает значение параметра "Expiration"
<a href="#">Magic</a>	Устанавливает значение параметра "Magic"
<b>Методы проверки необходимости открытия/разворота/закрытия позиций</b>	
virtual <a href="#">CheckOpenLong</a>	Определяет необходимость открытия длинной позиции
virtual <a href="#">CheckCloseLong</a>	Определяет необходимость закрытия длинной позиции
virtual <a href="#">CheckOpenShort</a>	Определяет необходимость открытия короткой позиции
virtual <a href="#">CheckCloseShort</a>	Определяет необходимость закрытия короткой позиции
virtual <a href="#">CheckReverseLong</a>	Определяет необходимость разворота длинной позиции
virtual <a href="#">CheckReverseShort</a>	Определяет необходимость разворота короткой позиции
<b>Методы установки параметров</b>	

virtual <a href="#">OpenLongParams</a>	Устанавливает параметры открытия длинной позиции
virtual <a href="#">OpenShortParams</a>	Устанавливает параметры открытия короткой позиции
virtual <a href="#">CloseLongParams</a>	Устанавливает параметры закрытия длинной позиции
virtual <a href="#">CloseShortParams</a>	Устанавливает параметры закрытия короткой позиции
<b>Методы проверки необходимости управления отложенными ордерами</b>	
virtual <a href="#">CheckTrailingOrderLong</a>	Определяет необходимость модификации параметров отложенного ордера на покупку
virtual <a href="#">CheckTrailingOrderShort</a>	Определяет необходимость модификации параметров отложенного ордера на продажу
<b>Методы проверки формирования рыночных моделей</b>	
virtual <a href="#">LongCondition</a>	Возвращает результат проверки возникновения условия на покупку
virtual <a href="#">ShortCondition</a>	Возвращает результат проверки возникновения условия на продажу
virtual <a href="#">Direction</a>	Возвращает значение "взвешенного" направления движения цены

**Методы унаследованные от CObject**Prev, Prev, Next, [Save](#), [Load](#), [Type](#), [Compare](#)**Методы унаследованные от CExpertBase**
[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#)

## BasePrice

Устанавливает базовый уровень цены.

```
void BasePrice(
    double     value          // значение
)
```

### Параметры

*value*

[in] Новое значение параметра.

### Возвращаемое значение

Нет.

## UsedSeries

Получает флаги используемых таймсерий.

```
int UsedSeries()
```

### Возвращаемое значение

Флаги используемых таймсерий (если не установлен другой символ или другой таймфрейм), иначе 0.

## Weight

Устанавливает значение параметра "Weight".

```
void Weight(
    double    value          // значение
)
```

### Параметры

*value*

[in] Параметр "Weight".

### Возвращаемое значение

Нет.

## PatternUsage

Устанавливает значение параметра "PatternsUsage".

```
void PatternUsage(
    double    value          // значение
)
```

### Параметры

*value*

[in] Новое значение параметра "PatternsUsage".

### Возвращаемое значение

Нет.

## General

Устанавливает значение параметра "General".

```
void General(
    int    value          // значение
)
```

### Параметры

*value*

[in] Новое значение параметра "General".

### Возвращаемое значение

Нет.

## Ignore

Устанавливает значение параметра "Ignore".

```
void Ignore(
    long    value          // значение
)
```

### Параметры

*value*

[in] Новое значение параметра "Ignore".

### Возвращаемое значение

Нет.

## Invert

Устанавливает значение параметра "Invert".

```
void Invert(  
    long value // значение  
)
```

### Параметры

*value*

[in] Новое значение параметра "Invert".

### Возвращаемое значение

Нет.

## ThresholdOpen

Устанавливает значение параметра "ThresholdOpen".

```
void ThresholdOpen(
    long    value          // значение
)
```

### Параметры

*value*

[in] Новое значение параметра "ThresholdOpen".

### Возвращаемое значение

Нет.

### Примечание

Параметр "ThresholdOpen" может принимать значения от 0 до 100. Используется для определения необходимости открытия позиции по результатам "голосования".

## ThresholdClose

Устанавливает значение параметра "ThresholdClose".

```
void ThresholdOpen(
    long    value          // значение
)
```

### Параметры

*value*

[in] Новое значение параметра "ThresholdClose".

### Возвращаемое значение

Нет.

### Примечание

Параметр "ThresholdClose" может принимать значения от 0 до 100. Используется для определения необходимости закрытия позиции по результатам "голосования".

## PriceLevel

Устанавливает значение параметра "PriceLevel".

```
void PriceLevel(
    double    value          // значение
)
```

### Параметры

*value*

[in] Новое значение параметра "PriceLevel".

### Возвращаемое значение

Нет.

### Примечание

Параметр "PriceLevel" задается в единицах измерения ценовых уровней. Значение единицы измерения ценовых уровней возвращает метод PriceLevelUnit(). Используется для задания уровня открытия относительно базовой цены.

## StopLevel

Устанавливает значение параметра "StopLevel".

```
void StopLevel(
    double    value          // значение
)
```

### Параметры

*value*

[in] Новое значение параметра "StopLevel".

### Возвращаемое значение

Нет.

### Примечание

Параметр "StopLevel" задается в единицах измерения ценовых уровней. Значение единицы измерения ценовых уровней возвращает метод PriceLevelUnit(). Используется для задания уровня фиксации убытков относительно цены открытия.

## TakeLevel

Устанавливает значение параметра "TakeLevel".

```
void TakeLevel(
    double    value          // значение
)
```

### Параметры

*value*

[in] Новое значение параметра "TakeLevel".

### Возвращаемое значение

Нет.

### Примечание

Параметр "TakeLevel" задается в единицах измерения ценовых уровней. Значение единицы измерения ценовых уровней возвращает метод PriceLevelUnit(). Используется для задания уровня фиксации прибыли относительно цены открытия.

## Expiration

Устанавливает значение параметра "Expiration".

```
void Expiration(
    int    value          // значение
)
```

### Параметры

*value*

[in] Новое значение параметра "Expiration".

### Возвращаемое значение

Нет.

### Примечание

Параметр "Expiration" задается в барах. Используется при расчёте времени истечения отложенного ордера (если предполагается вход не по текущей рыночной цене).

## Magic

Устанавливает значение параметра "Magic".

```
void Magic(  
    int    value          // значение  
)
```

### Параметры

*value*

[in] Новое значение параметра "Magic" (идентификатор эксперта).

### Возвращаемое значение

Нет.

## ValidationSettings

Проверяет корректность настроек объекта.

```
virtual bool ValidationSettings()
```

### Возвращаемое значение

true - если настройки объекта корректны, иначе - false.

## InitIndicators

Инициализирует необходимые индикаторы и таймсерии.

```
virtual bool InitIndicators(
    CIndicators* indicators // указатель
)
```

### Параметры

*indicators*

[in] Указатель на объект-коллекцию индикаторов и таймсерий.

### Возвращаемое значение

true - в случае успешного завершения, иначе false.

### Примечание

Необходимые таймсерии инициализируются в том случае, если объект использует инструмент или таймфрейм, отличный от установленного при начальной инициализации.

## AddFilter

Добавляет фильтр в комбинированный сигнал.

```
virtual bool AddFilter(  
    CExpertSignal* filter // указатель  
)
```

### Параметры

*filter*

[in] Указатель на объект-фильтр.

### Возвращаемое значение

true - в случае успешного завершения, иначе false.

## CheckOpenLong

Определяет необходимость открытия длинной позиции.

```
virtual bool CheckOpenLong(
    double& price,           // ссылка
    double& sl,              // ссылка
    double& tp,              // ссылка
    datetime& expiration     // ссылка
)
```

### Параметры

*price*

[in][out] Ссылка на переменную для размещения цены открытия.

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

*expiration*

[in][out] Ссылка на переменную для размещения времени истечения ордера (в случае необходимости).

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

## CheckOpenShort

Определяет необходимость открытия короткой позиции.

```
virtual bool CheckOpenShort(
    double& price,           // ссылка
    double& sl,              // ссылка
    double& tp,              // ссылка
    datetime& expiration     // ссылка
)
```

### Параметры

*price*

[in][out] Ссылка на переменную для размещения цены открытия.

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

*expiration*

[in][out] Ссылка на переменную для размещения времени истечения ордера (в случае необходимости).

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

## OpenLongParams

Устанавливает параметры открытия длинной позиции.

```
virtual bool OpenLongParams (
    double& price,           // ссылка
    double& sl,              // ссылка
    double& tp,              // ссылка
    datetime& expiration     // ссылка
)
```

### Параметры

*price*

[in][out] Ссылка на переменную для размещения цены открытия.

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

*expiration*

[in][out] Ссылка на переменную для размещения времени истечения ордера (в случае необходимости).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OpenShortParams

Устанавливает параметры открытия короткой позиции.

```
virtual bool OpenShortParams(
    double& price,           // ссылка
    double& sl,              // ссылка
    double& tp,              // ссылка
    datetime& expiration     // ссылка
)
```

### Параметры

*price*

[in][out] Ссылка на переменную для размещения цены открытия.

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

*expiration*

[in][out] Ссылка на переменную для размещения времени истечения ордера (в случае необходимости).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CheckCloseLong

Определяет необходимость закрытия длинной позиции.

```
virtual bool CheckCloseLong(
    double& price          // ссылка
)
```

### Параметры

*price*

[in][out] Ссылка на переменную для размещения цены закрытия.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

## CheckCloseShort

Определяет необходимость закрытия короткой позиции.

```
virtual bool CheckCloseShort(
    double& price          // ссылка
)
```

### Параметры

*price*

[in][out] Ссылка на переменную для размещения цены закрытия.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

## CloseLongParams

Устанавливает параметры закрытия длинной позиции.

```
virtual bool CloseLongParams(  
    double& price // ссылка  
)
```

### Параметры

*price*

[in][out] Ссылка на переменную для размещения цены закрытия.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CloseShortParams

Устанавливает параметры закрытия короткой позиции.

```
virtual bool CloseShortParams(
    double& price           // ссылка
)
```

### Параметры

*price*

[in][out] Ссылка на переменную для размещения цены закрытия.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CheckReverseLong

Определяет необходимость разворота длинной позиции.

```
virtual bool CheckReverseLong(
    double& price,           // ссылка
    double& sl,              // ссылка
    double& tp,              // ссылка
    datetime& expiration     // ссылка
)
```

### Параметры

*price*

[in][out] Ссылка на переменную для размещения цены разворота.

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

*expiration*

[in][out] Ссылка на переменную для размещения времени истечения ордера (в случае необходимости).

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

## CheckReverseShort

Определяет необходимость разворота короткой позиции.

```
virtual bool CheckReverseShort(
    double& price,           // ссылка
    double& sl,              // ссылка
    double& tp,              // ссылка
    datetime& expiration     // ссылка
)
```

### Параметры

*price*

[in][out] Ссылка на переменную для размещения цены разворота.

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

*expiration*

[in][out] Ссылка на переменную для размещения времени истечения ордера (в случае необходимости).

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

## CheckTrailingOrderLong

Определяет необходимость модификации параметров отложенного ордера на покупку.

```
virtual bool CheckTrailingOrderLong(
    COrderInfo*     order,           // указатель
    double&         price          // ссылка
)
```

### Параметры

*order*

[in] Указатель на объект [COrderInfo](#).

*price*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

## CheckTrailingOrderShort

Определяет необходимость модификации параметров отложенного ордера на продажу.

```
virtual bool CheckTrailingOrderShort(
    COrderInfo*     order,           // указатель
    double&         price           // ссылка
)
```

### Параметры

*order*

[in] Указатель на объект [COrderInfo](#).

*price*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

## LongCondition

Возвращает результат проверки возникновения условия на покупку.

```
virtual int LongCondition()
```

### Возвращаемое значение

В случае возникновения условия на покупку, возвращает число от 1 до 100 (чем больше значение, тем больше "сила" сигнала), в случае отсутствия сигнала, возвращается 0.

### Примечание

Базовый класс не имеет реализацию алгоритма определения условия на покупку, поэтому метод базового класса всегда возвращает 0.

## ShortCondition

Возвращает результат проверки возникновения условия на продажу.

```
virtual int ShortCondition()
```

### Возвращаемое значение

В случае возникновения условия на продажу, возвращает число от 1 до 100 (чем больше значение, тем больше "сила" сигнала), в случае отсутствия сигнала, возвращается 0.

### Примечание

Базовый класс не имеет реализацию алгоритма определения условия на продажу, поэтому метод базового класса всегда возвращает 0.

## Direction

Возвращает значение "взвешенного" направления движения цены.

```
virtual double Direction()
```

### Возвращаемое значение

В случае вероятного движения цены вверх, значение будет  $>0$ , в случае вероятного движения цены вниз, значение будет  $<0$ . Чем больше абсолютное значение, тем "сильнее" сигнал.

### Примечание

При наличии встроенных фильтров, их результат будет учтен при определении общего направления.

## Класс CExpertTrailing

Класс `CExpertTrailing` является базовым классом для реализации алгоритмов сопровождения открытых позиций, поэтому он, предоставляя интерфейсы, сам ничего не делает.

Для того чтобы "трейлинг" заработал по-другому, нужно:

1. Определиться с алгоритмами сопровождения открытых позиций;
2. Создать свой класс, унаследовав его от `CExpertTrailing`;
3. Переопределить в своем классе виртуальные методы базового, заложив в них соответствующие алгоритмы.

В качестве примера можно рассмотреть любой mqh-файл из папки `Expert\Trailing\`.

### Описание

Класс `CExpertTrailing` является основой для реализации алгоритмов сопровождения открытых позиций.

### Декларация

```
class CExpertTrailing : public CExpertBase
```

### Заголовок

```
#include <Expert\ExpertTrailing.mqh>
```

### Иерархия наследования

```
CObject  
CExpertBase  
CExpertTrailing
```

### Прямые потомки

[CTrailingFixedPips](#), [CTrailingMA](#), [CTrailingNone](#), [CTrailingPSAR](#)

### Методы класса по группам

Методы проверки необходимости управления отложенными ордерами	
virtual <a href="#">CheckTrailingStopLong</a>	Определяет необходимость модификации параметров длинной позиции
virtual <a href="#">CheckTrailingStopShort</a>	Определяет необходимость модификации параметров короткой позиции

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Методы унаследованные от CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [ValidationSettings](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

## CheckTrailingStopLong

Определяет необходимость модификации параметров длинной позиции.

```
virtual bool CheckTrailingStopLong(
    CPositionInfo* position,           // указатель
    double&          sl,              // ссылка для Stop Loss
    double&          tp               // ссылка для Take Profit
)
```

### Параметры

*position*

[in] Указатель на объект [CPositionInfo](#).

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

### Примечание

Метод базового класса всегда возвращает false.

## CheckTrailingStopShort

Определяет необходимость модификации параметров короткой позиции.

```
virtual bool CheckTrailingStopShort(
    CPositionInfo* position,           // указатель
    double&          sl,              // ссылка для Stop Loss
    double&          tp               // ссылка для Take Profit
)
```

### Параметры

*position*

[in] Указатель на объект [CPositionInfo](#).

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

### Примечание

Метод базового класса всегда возвращает false.

## Класс CExpertMoney

Класс CExpertMoney является базовым классом для реализации алгоритмов управления капиталом и рисками.

### Описание

Класс CExpertMoney является основой для реализации алгоритмов управления капиталом и рисками.

### Декларация

```
class CExpertMoney : public CObject
```

### Заголовок

```
#include <Expert\ExpertMoney.mqh>
```

### Иерархия наследования

```
CObject  
CExpertBase  
CExpertMoney
```

### Прямые потомки

[CMoneyFixedLot](#), [CMoneyFixedMargin](#), [CMoneyFixedRisk](#), [CMoneyNone](#), [CMoneySizeOptimized](#)

### Методы класса по группам

Доступ к защищенным данным	
<a href="#">Percent</a>	Устанавливает значение параметра "Процент риска"
<a href="#">Инициализация</a>	
<a href="#">virtual ValidationSettings</a>	Проверяет корректность настроек
<a href="#">Методы проверки необходимости открытия/разворота/закрытия позиций</a>	
<a href="#">virtual CheckOpenLong</a>	Определяет объем для открытия длинной позиции
<a href="#">virtual CheckOpenShort</a>	Определяет объем для открытия короткой позиции
<a href="#">virtual CheckReverse</a>	Определяет объем для разворота позиции
<a href="#">virtual CheckClose</a>	Определяет необходимость закрытия позиции

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

**Методы унаследованные от CExpertBase**

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#),  
[RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#),  
[InitIndicators](#)

## Percent

Устанавливает процент риска.

```
void Percent(
    double percent           // процент риска
)
```

### Параметры

*percent*

[in] Процент риска.

### Возвращаемое значение

Нет.

## ValidationSettings

Проверяет корректность настроек.

```
virtual bool ValidationSettings()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Метод базового класса всегда возвращает true.

## CheckOpenLong

Определяет объем для открытия длинной позиции.

```
virtual double CheckOpenLong(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Цена открытия длинной позиции.

*sl*

[in] Цена Stop Loss длинной позиции.

### Возвращаемое значение

Объем для открытия длинной позиции.

## CheckOpenShort

Определяет объем для открытия короткой позиции.

```
virtual double CheckOpenShort(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Цена открытия короткой позиции.

*sl*

[in] Цена Stop Loss короткой позиции.

### Возвращаемое значение

Объем для открытия короткой позиции.

## CheckReverse

Определяет объем для разворота позиции.

```
virtual double CheckReverse(
    CPositionInfo* position,           // указатель
    double          sl                 // цена Stop Loss
)
```

### Параметры

*position*

[in] Указатель на объект [CPositionInfo](#).

*sl*

[in] Цена Stop Loss позиции.

### Возвращаемое значение

Объем для разворота позиции.

## CheckClose

Определяет необходимость закрытия позиции.

```
virtual double CheckClose()
```

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

## Модули торговых сигналов

В стандартную поставку клиентского терминала входит набор модулей торговых сигналов для "Мастера MQL5". При создании эксперта в Мастере MQL5, вы можете включить в него любую комбинацию модулей торговых сигналов (до 64). Финальное решение о совершении торговой операции принимается на основе совокупного анализа сигналов всех включенных модулей. Подробное описание механизма принятия решений приведено [ниже](#).

В стандартную поставку входят следующие модули сигналов:

- [Сигналы индикатора Accelerator Oscillator](#)
- [Сигналы индикатора Adaptive Moving Average](#)
- [Сигналы индикатора Awesome Oscillator](#)
- [Сигналы осциллятора Bears Power](#)
- [Сигналы осциллятора Bulls Power](#)
- [Сигналы осциллятора Commodity Channel Index](#)
- [Сигналы осциллятора DeMarker](#)
- [Сигналы индикатора Double Exponential Moving Average](#)
- [Сигналы индикатора Envelopes](#)
- [Сигналы индикатора Fractal Adaptive Moving Average](#)
- [Сигналы внутридневного временного фильтра](#)
- [Сигналы осциллятора MACD](#)
- [Сигналы индикатора Moving Average](#)
- [Сигналы индикатора Parabolic SAR](#)
- [Сигналы осциллятора Relative Strength Index](#)
- [Сигналы осциллятора Relative Vigor Index](#)
- [Сигналы осциллятора Stochastic](#)
- [Сигналы осциллятора Triple Exponential Average](#)
- [Сигналы индикатора Triple Exponential Moving Average](#)
- [Сигналы осциллятора Williams Percent Range](#)

## Механизм принятия торговых решений на основе модулей сигналов

Механизм принятия торговых решений можно представить в виде следующих основных положений:

- Каждый из модулей сигналов обладает своим набором рыночных моделей (определенное сочетание цен и значений индикатора).
- Каждой рыночной модели установлена значимость, измеряемая от 1 до 100. Чем больше значение, тем сильнее модель.
- Каждая из моделей генерирует прогноз движения цены в определенном направлении.
- Прогноз модуля сигналов является результатом поиска заложенных моделей и выдается в виде числа в диапазоне от -100 до +100, где знак определяет направление предполагаемого

движения (отрицательный – цена будет падать, положительный – цена будет расти). Абсолютное значение соответствует силе найденной наилучшей модели.

- Прогноз каждого модуля отправляется на голосование с весовым коэффициентом от 0 до 1.0, указанным в его настройках ("Weight").
- Итогом голосования является число от -100 до +100, где знак определяет направление прогнозируемого движения, а абсолютное значение характеризует силу сигнала. Оно вычисляется как среднеарифметическое взвешенных прогнозов всех модулей сигналов. Данное итоговое значение используется в советнике для принятия торговых решений.

В настройках каждого сгенерированного эксперта присутствуют два параметра – пороговые значения для принятия решения об открытии или закрытии позиции (ThresholdOpen и ThresholdClose), – которые могут иметь значение от 0 до 100. Если сила итогового сигнала (абсолютная величина) преодолевает пороговое значение, принимается решение о совершении торговой операции в направлении, соответствующему знаку прогноза.

## Примеры

Пусть существует некий советник с пороговыми значениями ThresholdOpen=20 и ThresholdClose=90. В принятии решений о торговых операциях участвуют модули сигналов на основе [MA](#) с весом 0.4 и [Stochastic](#) с весом 0.8. Рассмотрим два варианта полученных торговых сигналов:

### Вариант 1.

Цена пересекла снизу вверх восходящий индикатор MA. Это соответствует одной из заложенных в [модуле MA](#) рыночной модели, предполагающей рост цены. Ее значимость равняется 100. В это же время осциллятор Stochastic развернулся вниз и сформировал дивергенцию с ценой. Это является одной из заложенных в [модуле Stochastic](#) моделей, предполагающей падение цены. Значимость этой модели равна 80.

Рассчитаем результат итогового голосования. Взвешенный прогноз, полученный от модуля MA, рассчитывается как  $0.4 * 100 = 40$ . Взвешенный прогноз от модуля Stochastic рассчитывается как  $0.8 * (-80) = -64$ . Итоговый прогноз вычисляется нахождением среднеарифметического этих двух взвешенных прогнозов:  $(40 - 64)/2 = -12$ . Это является сигналом на продажу с условной силой 12. Пороговое значение, равное 20, не достигнуто. Соответственно торговая операция не совершается.

### Вариант 2.

Цена пересекла сверху вниз восходящий индикатор MA. Это соответствует одной из заложенных в [модуле MA](#) рыночной модели, предполагающей рост цены. Ее значимость равняется 10. В это же время осциллятор Stochastic развернулся вниз и сформировал дивергенцию с ценой. Это является одной из заложенных в [модуле Stochastic](#) моделей, предполагающей падение цены. Значимость этой модели равна 80.

Рассчитаем результат итогового голосования. Взвешенный прогноз, полученный от модуля MA, рассчитывается как  $0.4 * 10 = 4$ . Взвешенный прогноз от модуля Stochastic рассчитывается как  $0.8 * (-80) = -64$ . Итоговый прогноз вычисляется нахождением среднеарифметического этих двух взвешенных прогнозов:  $(4 - 64)/2 = -30$ . Это является сигналом на продажу с условной силой 30. Пороговое значение, равное 20, достигнуто. Соответственно, результатом является сигнал на открытие короткой позиции.



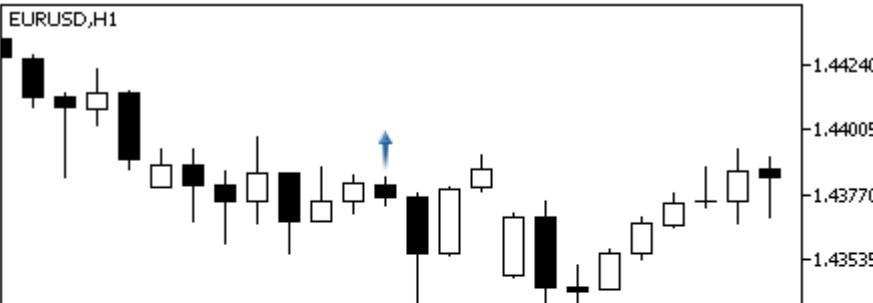
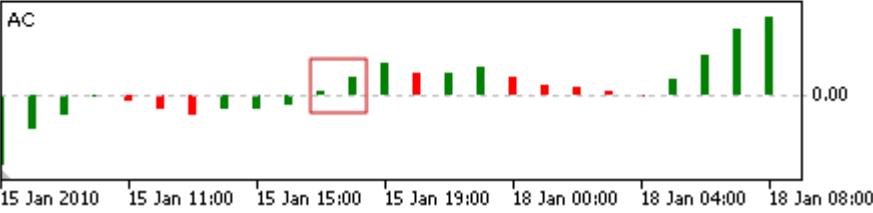
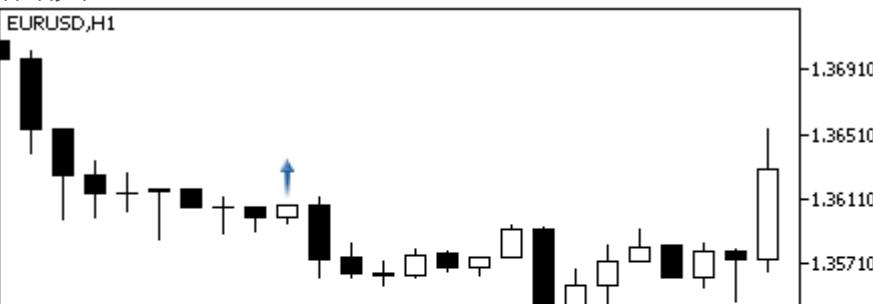
- a) Дивергенция цены и осциллятора Stochastic (вариант 1 и 2).
- б) Цена пересекла индикатор MA снизу вверх (вариант 1).
- в) Цена пересекла индикатор MA сверху вниз (вариант 2).

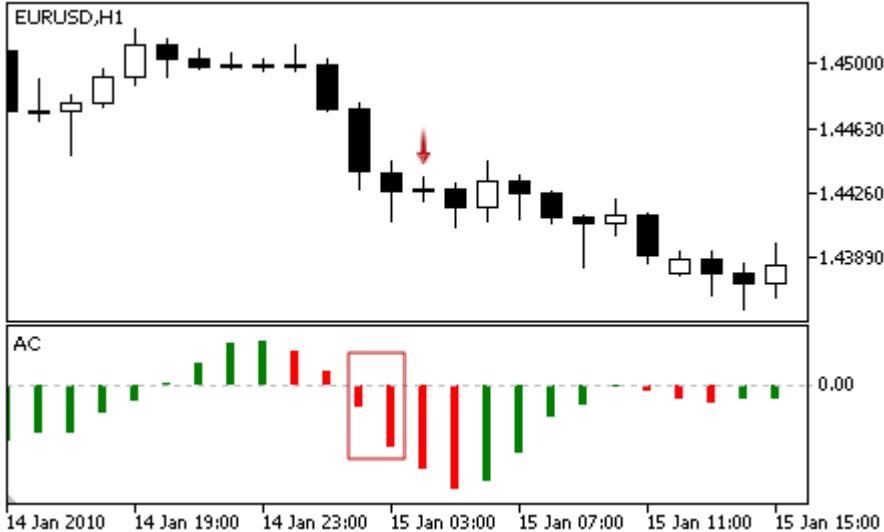
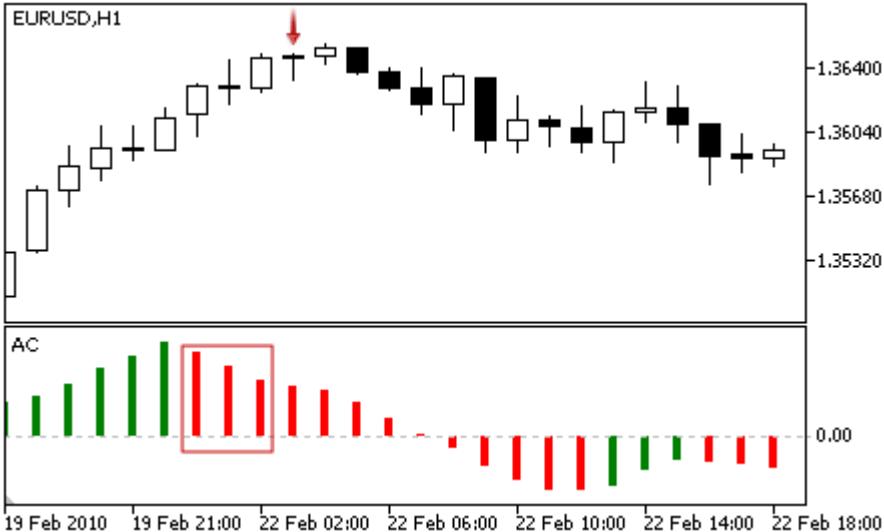
## Сигналы индикатора Accelerator Oscillator

Данный модуль сигналов основан на рыночных моделях индикатора [Accelerator Oscillator](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li>Значение индикатора выше 0 и оно растет на анализируемом баре и предыдущем.</li> </ul>   <p>15 Jan 2010 15 Jan 11:00 15 Jan 15:00 15 Jan 19:00 18 Jan 00:00 18 Jan 04:00 18 Jan 08:00</p> <ul style="list-style-type: none"> <li>Значение индикатора ниже 0 и оно растет на анализируемом баре и двух предыдущих.</li> </ul>   <p>17 Feb 2010 17 Feb 19:00 17 Feb 23:00 18 Feb 03:00 18 Feb 07:00 18 Feb 11:00 18 Feb 15:00</p>
За продажу	<ul style="list-style-type: none"> <li>Значение индикатора ниже 0 и оно падает на анализируемом баре и предыдущем.</li> </ul>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Значение индикатора выше 0 и оно падает на анализируемом баре и двух предыдущих.</li> </ul> 
Не против покупки	Значение индикатора на анализируемом баре растет.
Не против продажи	Значение индикатора на анализируемом баре падает.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

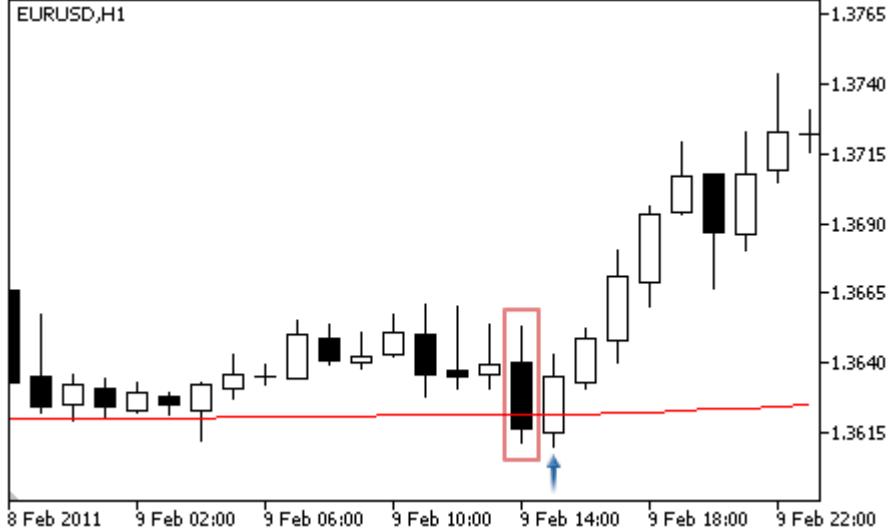
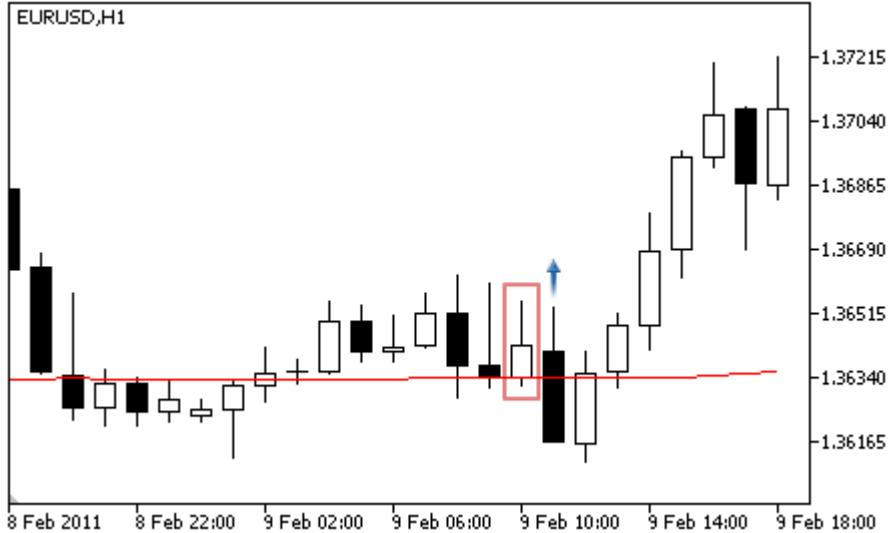
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.

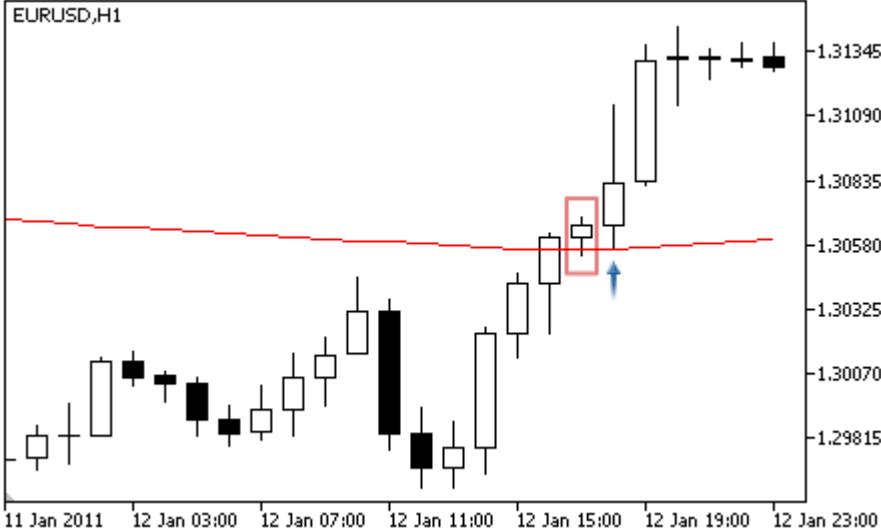
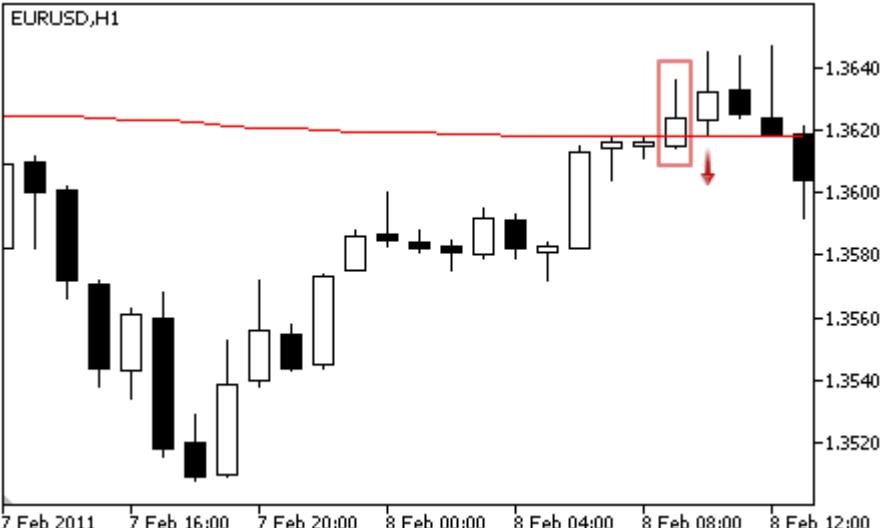
## Сигналы индикатора Adaptive Moving Average

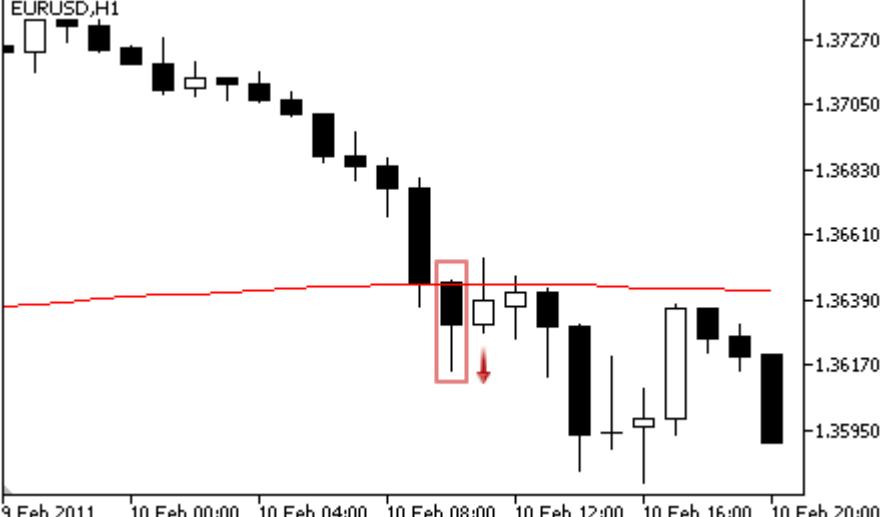
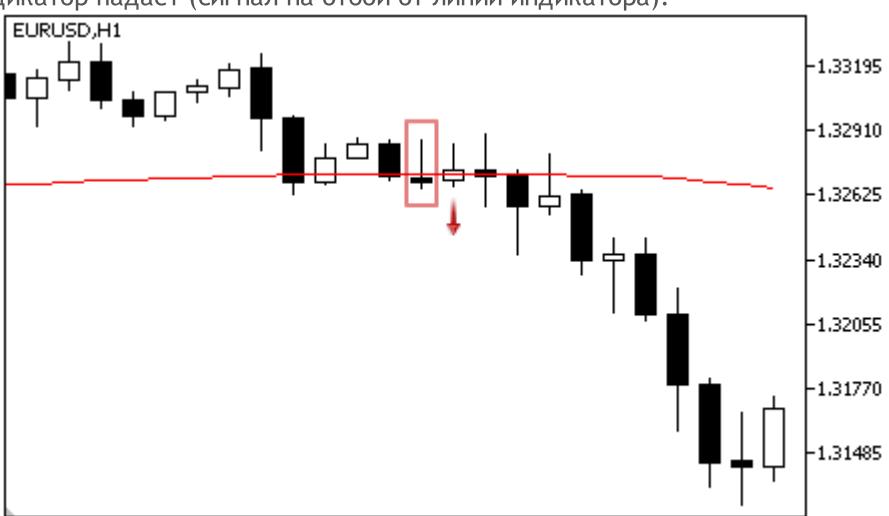
Данный модуль сигналов основан на рыночных моделях индикатора [Adaptive Moving Average](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li>• Несформировавшийся прокол. Цена пересекла индикатор сверху вниз(цена Open анализируемого бара выше линии индикатора, а цена Close - ниже), но индикатор растет (слабый сигнал на отбой от линии индикатора).</li> </ul> 
	<ul style="list-style-type: none"> <li>• Пересечение скользящей средней. Цена пересекла индикатор снизу вверх (цена Open анализируемого бара ниже линии индикатора, а цена Close - выше) и индикатор растет (сильный сигнал).</li> </ul> 

Тип сигнала	Описание условий
	<ul style="list-style-type: none"> <li>Сформировавшийся прокол. Цена пересекла индикатор нижней тенью (цены Open и Close анализируемого бара выше линии индикатора, а цена Low ниже) и индикатор растет (сигнал на отбой от линии индикатора).</li> </ul>  <p>EURUSD,H1 11 Jan 2011 12 Jan 03:00 12 Jan 07:00 12 Jan 11:00 12 Jan 15:00 12 Jan 19:00 12 Jan 23:00</p>
За продажу	<ul style="list-style-type: none"> <li>Несформировавшийся прокол. Цена пересекла индикатор снизу вверх (цена Open анализируемого бара ниже линии индикатора, а цена Close - выше), но индикатор падает (слабый сигнал на отбой от линии индикатора).</li> </ul>  <p>EURUSD,H1 7 Feb 2011 7 Feb 16:00 7 Feb 20:00 8 Feb 00:00 8 Feb 04:00 8 Feb 08:00 8 Feb 12:00</p> <ul style="list-style-type: none"> <li>Пересечение скользящей средней. Цена пересекла индикатор сверху вниз (цена Open анализируемого бара выше линии индикатора, а цена Close - ниже) и индикатор падает (сильный сигнал).</li> </ul>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Сформировавшийся прокол. Цена пересекла индикатор верхней тенью (цены Open и Close анализируемого бара ниже линии индикатора, а цена High выше) и индикатор падает (сигнал на отбой от линии индикатора).</li> </ul> 
Не против покупки	Цена находится выше индикатора.
Не против продажи	Цена находится ниже индикатора.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

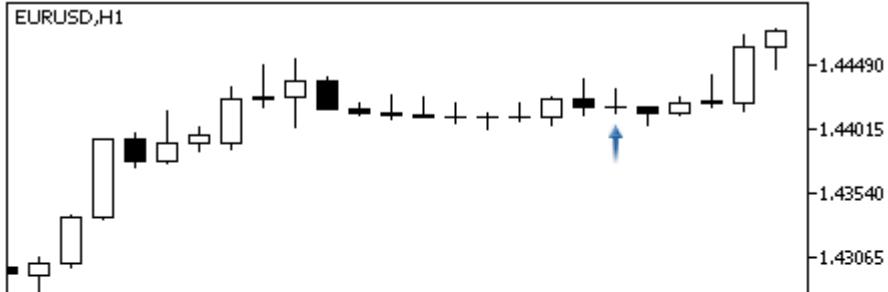
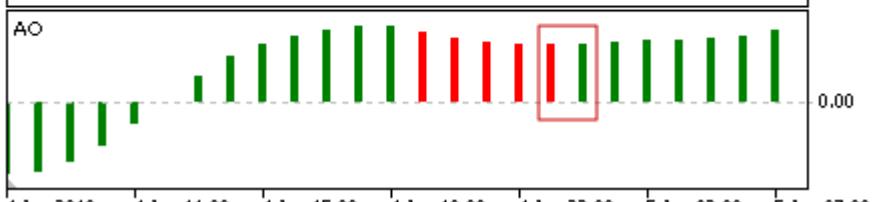
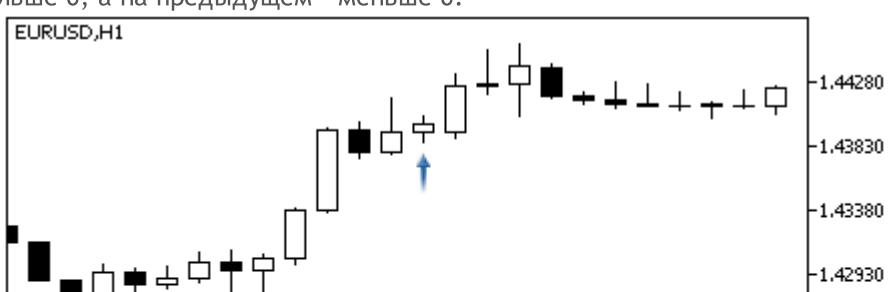
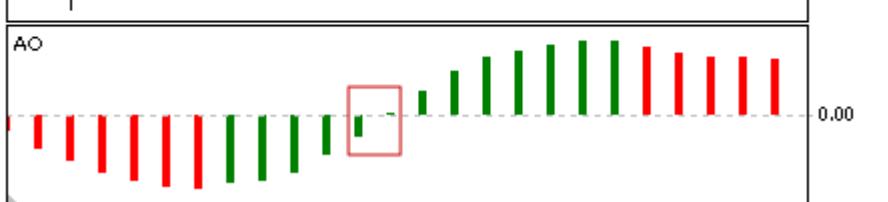
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodMA	Период усреднения индикатора.
Shift	Смещение индикатора по оси времени (в барах).
Method	<a href="#">Метод усреднения</a> .
Applied	<a href="#">Ценовой ряд</a> , на значениях которого рассчитывается индикатор.

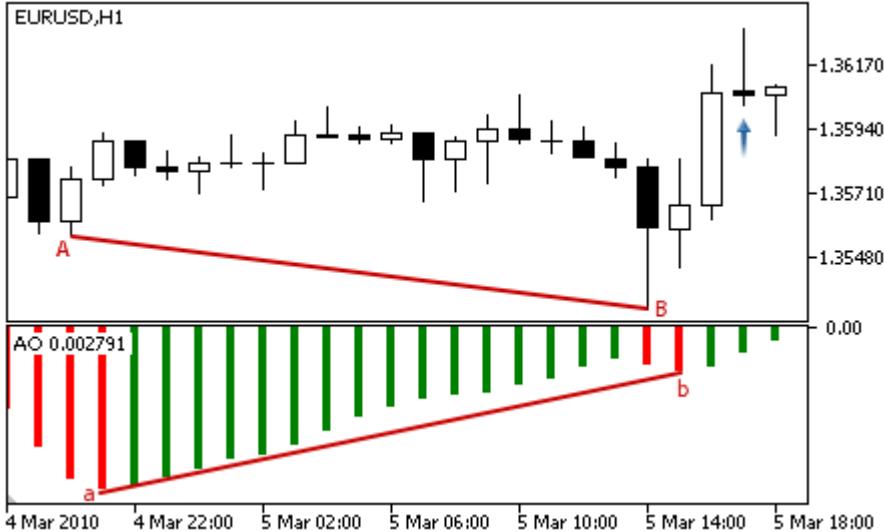
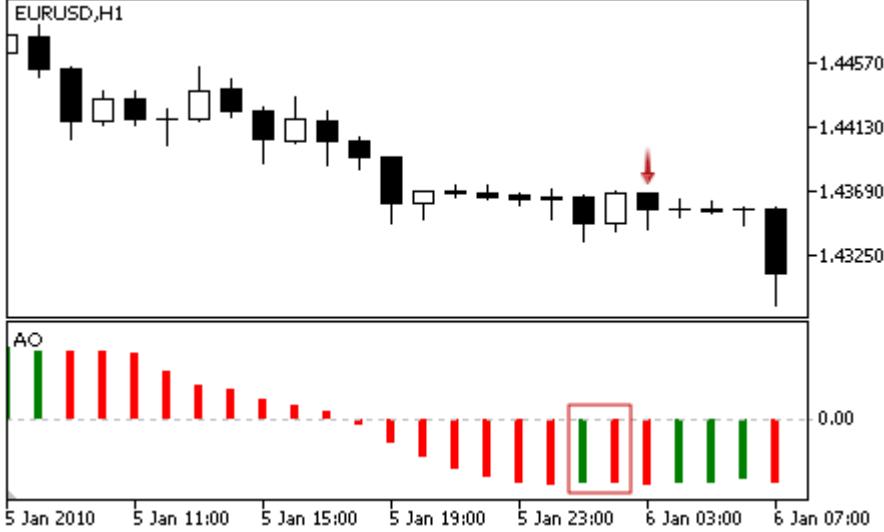
## Сигналы индикатора Awesome Oscillator

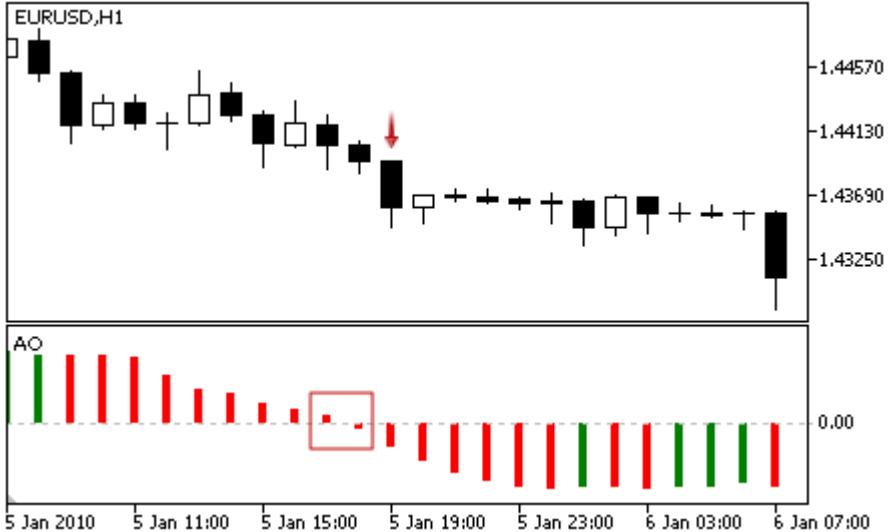
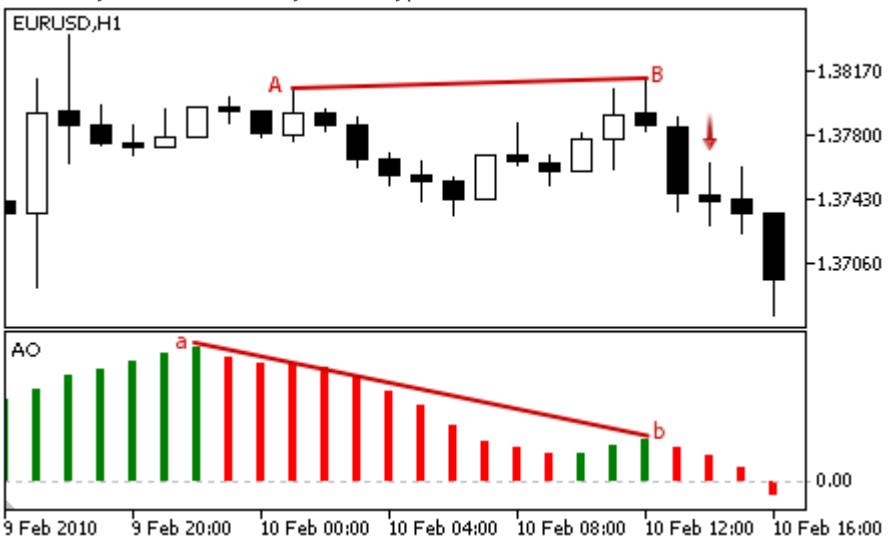
Данный модуль сигналов основан на рыночных моделях индикатора [Awesome Oscillator](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li><b>Блюдце</b> – значение индикатора на анализируемом баре растет, а на предыдущем оно падало, и при этом оба значения выше 0.</li> </ul>   <p>4 Jan 2010 4 Jan 11:00 4 Jan 15:00 4 Jan 19:00 4 Jan 23:00 5 Jan 03:00 5 Jan 07:00</p> <ul style="list-style-type: none"> <li><b>Пересечение нулевой линии</b> – значение индикатора на анализируемом баре больше 0, а на предыдущем - меньше 0.</li> </ul>   <p>4 Jan 2010 4 Jan 04:00 4 Jan 08:00 4 Jan 12:00 4 Jan 16:00 4 Jan 20:00 5 Jan 00:00</p> <ul style="list-style-type: none"> <li><b>Дивергенция</b> – первая анализируемая впадина индикатора мельче предыдущей, а соответствующая ей впадина цены глубже предыдущей. При этом индикатор не должен подниматься выше нулевого уровня.</li> </ul>
За продажу	

Тип сигнала	Описание условий
	 <p>EURUSD,H1</p> <p>AO 0.002791</p> <p>4 Mar 2010 4 Mar 22:00 5 Mar 02:00 5 Mar 06:00 5 Mar 10:00 5 Mar 14:00 5 Mar 18:00</p>
За продажу	<ul style="list-style-type: none"> <li>• Блюдце — значение индикатора на анализируемом баре падает, а на предыдущем оно росло, и при этом оба значения ниже 0.</li> </ul>  <p>EURUSD,H1</p> <p>AO</p> <p>5 Jan 2010 5 Jan 11:00 5 Jan 15:00 5 Jan 19:00 5 Jan 23:00 6 Jan 03:00 6 Jan 07:00</p> <ul style="list-style-type: none"> <li>• Пересечение нулевой линии — значение индикатора на анализируемом баре меньше 0, а на предыдущем - больше 0.</li> </ul>

Тип сигнала	Описание условий
	 <p>EURUSD,H1</p> <p>AO</p> <p>5 Jan 2010 5 Jan 11:00 5 Jan 15:00 5 Jan 19:00 5 Jan 23:00 6 Jan 03:00 6 Jan 07:00</p>
	<ul style="list-style-type: none"> <li>• Дивергенция – первый анализируемый пик индикатора ниже предыдущего, а соответствующий ему пик цены выше предыдущего. При этом индикатор не должен опускаться ниже нулевого уровня.</li> </ul>  <p>EURUSD,H1</p> <p>AO</p> <p>9 Feb 2010 9 Feb 20:00 10 Feb 00:00 10 Feb 04:00 10 Feb 08:00 10 Feb 12:00 10 Feb 16:00</p>
Не против покупки	Значение индикатора на анализируемом баре растет.
Не против продажи	Значение индикатора на анализируемом баре падает.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

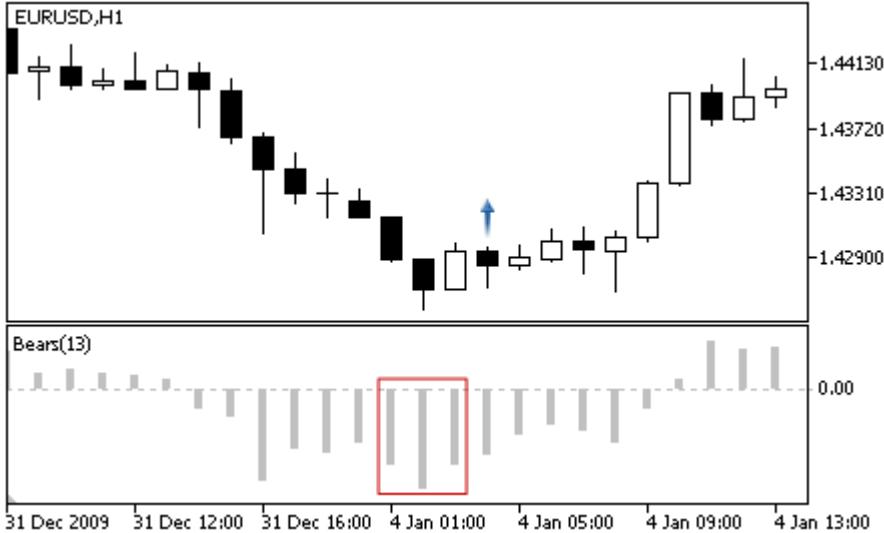
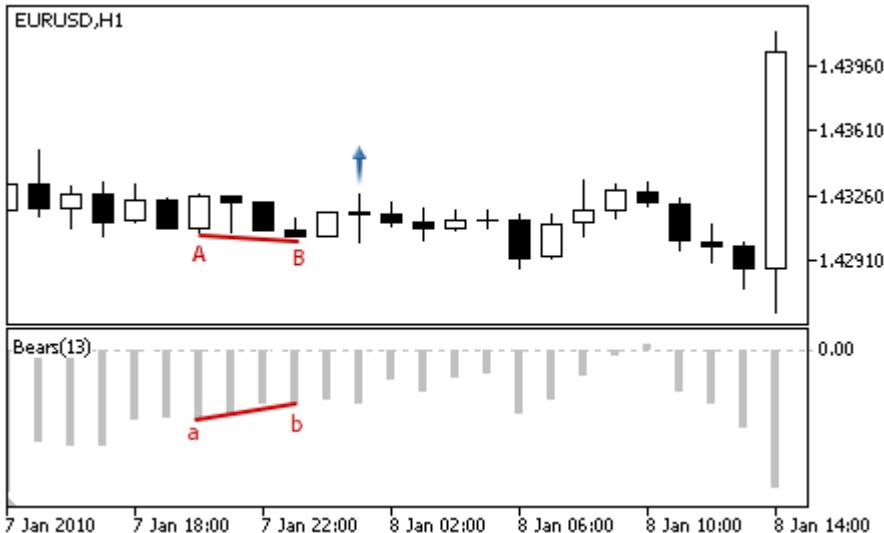
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.

## Сигналы осциллятора Bears Power

Данный модуль сигналов основан на рыночных моделях осциллятора [Bears Power](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li><b>Разворот</b> — осциллятор развернулся вверх и при этом его значение на анализируемом баре ниже 0.</li> </ul>  <p>EURUSD,H1</p> <p>Bears(13)</p> <p>31 Dec 2009 31 Dec 12:00 31 Dec 16:00 4 Jan 01:00 4 Jan 05:00 4 Jan 09:00 4 Jan 13:00</p> <ul style="list-style-type: none"> <li><b>Дивергенция</b> — первая анализируемая впадина осциллятора мельче предыдущей, а соответствующая ей впадина цены глубже предыдущей. При этом осциллятор не должен подниматься выше нулевого уровня.</li> </ul>  <p>EURUSD,H1</p> <p>Bears(13)</p> <p>7 Jan 2010 7 Jan 18:00 7 Jan 22:00 8 Jan 02:00 8 Jan 06:00 8 Jan 10:00 8 Jan 14:00</p>
За продажу	Сигналы для продажи отсутствуют.

Тип сигнала	Описание условий
Не против покупки	Значение осциллятора ниже 0.
Не против продажи	Сигналы отсутствуют.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

## Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

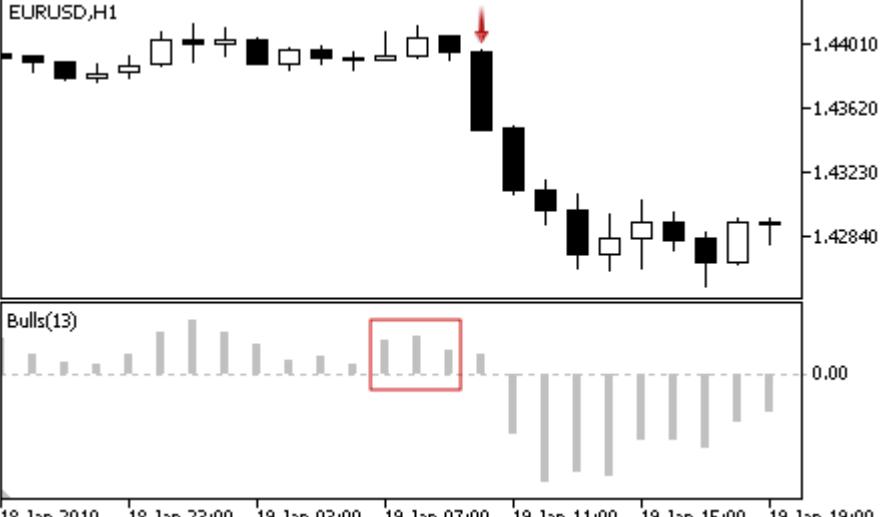
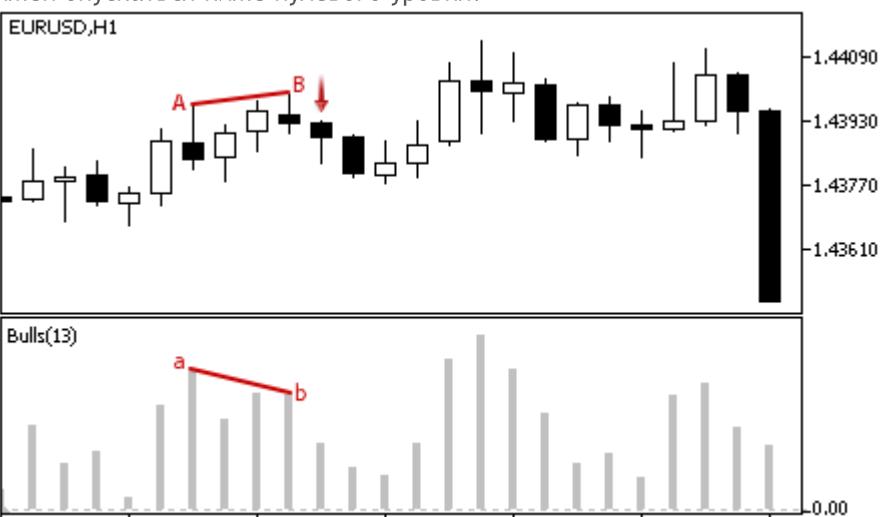
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodBears	Период расчета осциллятора.

## Сигналы индикатора Bulls Power

Данный модуль сигналов основан на рыночных моделях осциллятора [Bulls Power](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	Сигналы для покупки отсутствуют.
За продажу	<ul style="list-style-type: none"> <li><b>Разворот</b> — осциллятор развернулся вниз и при этом его значение на анализируемом баре выше 0.</li> </ul>  <p>EURUSD,H1 Bulls(13)</p> <p>18 Jan 2010 18 Jan 23:00 19 Jan 03:00 19 Jan 07:00 19 Jan 11:00 19 Jan 15:00 19 Jan 19:00</p> <ul style="list-style-type: none"> <li><b>Дивергенция</b> — первый анализируемый пик осциллятора ниже предыдущего, а соответствующий ему пик цены выше предыдущего. При этом осциллятор не должен опускаться ниже нулевого уровня.</li> </ul>  <p>EURUSD,H1 Bulls(13)</p> <p>18 Jan 2010 18 Jan 14:00 18 Jan 18:00 18 Jan 22:00 19 Jan 02:00 19 Jan 06:00 19 Jan 10:00</p>

Тип сигнала	Описание условий
Не против покупки	Сигналы отсутствуют.
Не против продажи	Значение осциллятора выше 0.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

## Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

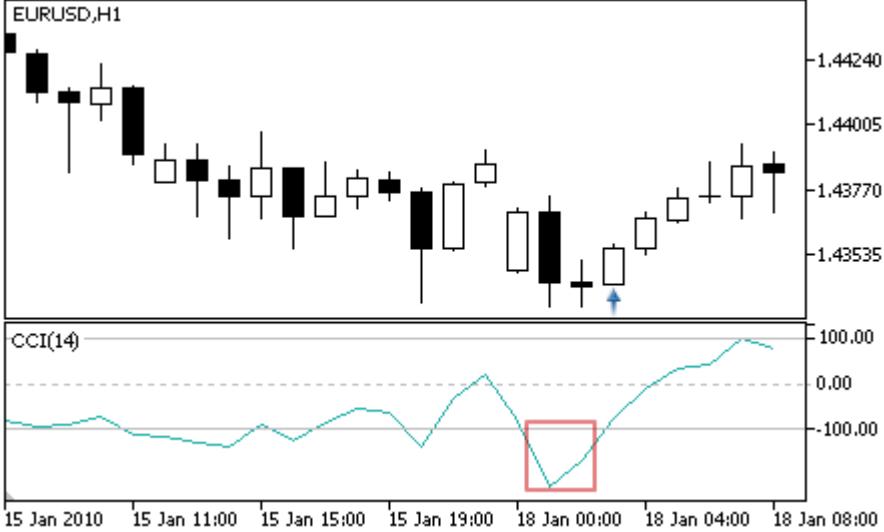
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodBulls	Период расчета осциллятора.

## Сигналы осциллятора Commodity Channel Index

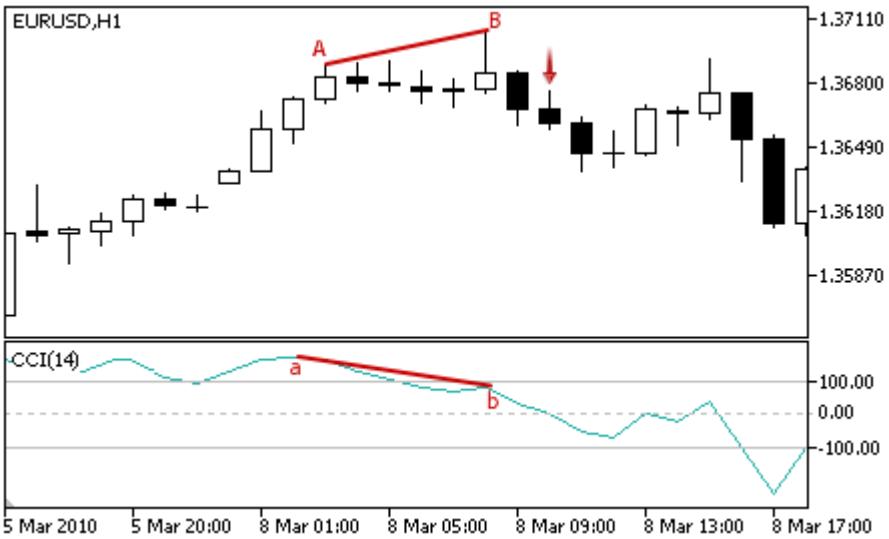
Данный модуль сигналов основан на рыночных моделях осциллятора [Commodity Channel Index](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li>Разворот за уровнем перепроданности — осциллятор развернулся вверх и его значение на анализируемом баре находится за уровнем перепроданности (по умолчанию -100).</li> </ul>  <p>EURUSD,H1</p> <p>CCI(14)</p> <p>15 Jan 2010 15 Jan 11:00 15 Jan 15:00 15 Jan 19:00 18 Jan 00:00 18 Jan 04:00 18 Jan 08:00</p> <ul style="list-style-type: none"> <li>Дивергенция — первая анализируемая впадина осциллятора мельче предыдущей, а соответствующая ей впадина цены глубже предыдущей.</li> </ul>  <p>EURUSD,H1</p> <p>CCI(14)</p> <p>31 Dec 2009 31 Dec 11:00 31 Dec 15:00 4 Jan 00:00 4 Jan 04:00 4 Jan 08:00 4 Jan 12:00</p>

Тип сигнала	Описание условий
	<ul style="list-style-type: none"> <li>Двойная дивергенция — осциллятор сформировал три последовательных впадины, каждая из которых мельче предыдущей, а цена сформировала три соответствующие им впадины, каждая из которых глубже предыдущей.</li> </ul> 
За продажу	<ul style="list-style-type: none"> <li>Разворот за уровнем перекупленности — осциллятор развернулся вниз и его значение на анализируемом баре находится за уровнем перекупленности (по умолчанию 100).</li> </ul>  <ul style="list-style-type: none"> <li>Дивергенция — первый анализируемый пик осциллятора ниже предыдущего, а соответствующий ему пик цены выше предыдущего.</li> </ul>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Двойная дивергенция – осциллятор сформировал три последовательных пика, каждый из которых ниже предыдущего, а цена сформировала три соответствующих им пика, каждый из которых выше предыдущего.</li> </ul> 
Не против покупки	Значение осциллятора на анализируемом баре растет.
Не против продажи	Значение осциллятора на анализируемом баре падает.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

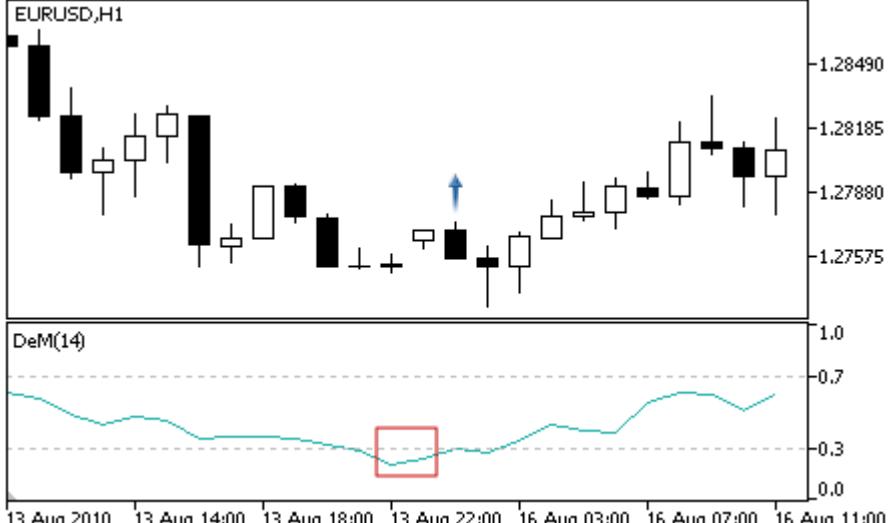
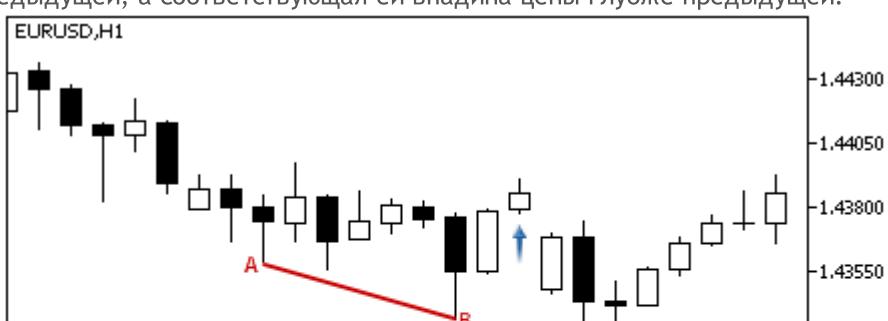
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodCCI	Период расчета осциллятора.
Applied	<a href="#">Ценовой ряд</a> , на значениях которого рассчитывается осциллятор.

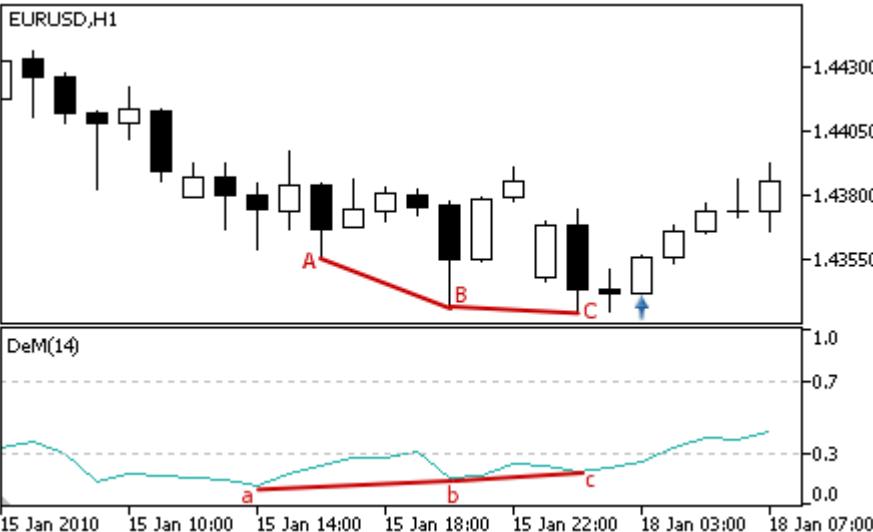
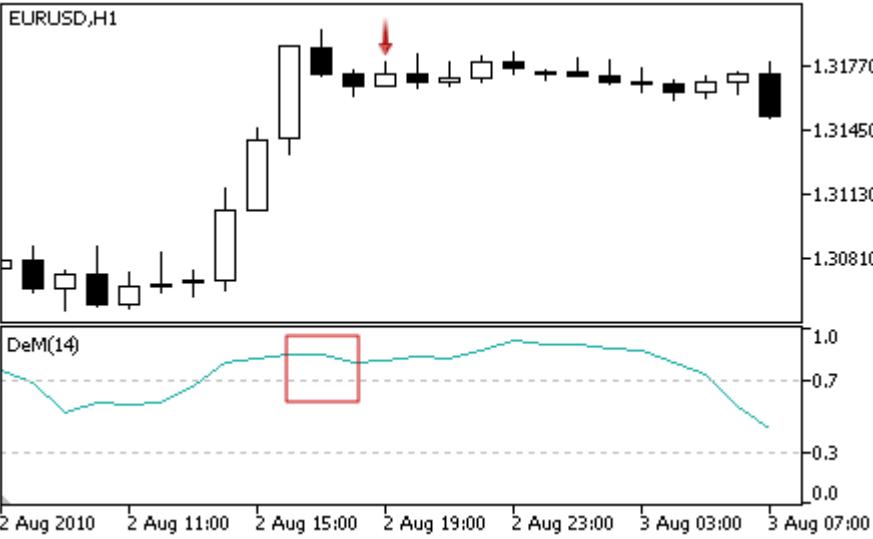
## Сигналы индикатора DeMarker

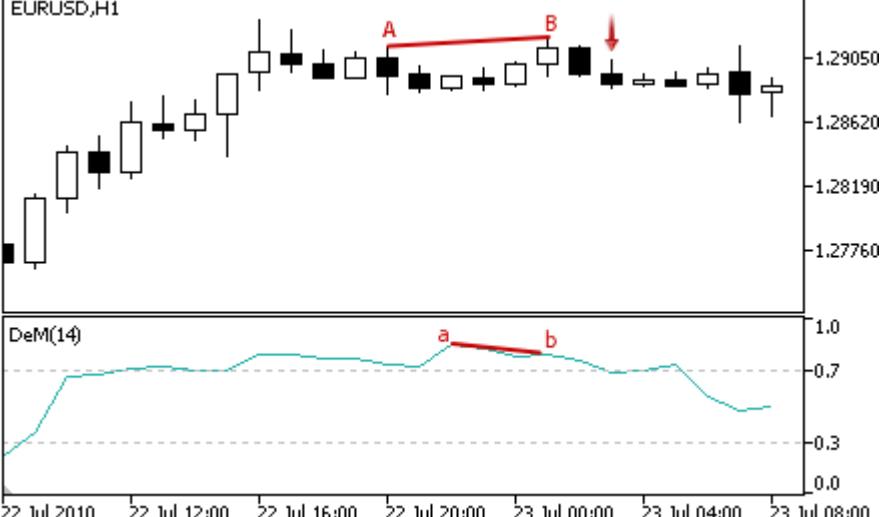
Данный модуль сигналов основан на рыночных моделях осциллятора [DeMarker](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li>Разворот за уровнем перепроданности – осциллятор развернулся вверх и его значение на анализируемом баре находится за уровнем перепроданности (по умолчанию 0.3).</li> </ul>  <p>The chart shows EURUSD price action on a H1 timeframe. The DeMarker indicator (DeM(14)) is plotted below the candles. A red box highlights a point where the DeM value has crossed above the 0.3 threshold, indicating an overbought condition. A blue arrow points upwards from this point, signifying a buy signal.</p>
	<ul style="list-style-type: none"> <li>Дивергенция – первая анализируемая впадина осциллятора мельче предыдущей, а соответствующая ей впадина цены глубже предыдущей.</li> </ul>  <p>The chart shows EURUSD price action on a H1 timeframe. The DeMarker indicator (DeM(14)) is plotted below the candles. A red trendline connects two consecutive troughs of the DeMarker line, labeled 'A' and 'B'. Trough 'A' is shallower than trough 'B'. The corresponding price trough 'a' is deeper than the previous one, illustrating a divergence pattern.</p>

Тип сигнала	Описание условий
	<ul style="list-style-type: none"> <li>Двойная дивергенция – осциллятор сформировал три последовательных впадины, каждая из которых мельче предыдущей, а цена сформировала три соответствующие им впадины, каждая из которых глубже предыдущей.</li> </ul> 
За продажу	<ul style="list-style-type: none"> <li>Разворот за уровнем перекупленности – осциллятор развернулся вниз и его значение на анализируемом баре находится за уровнем перекупленности (по умолчанию 0.7).</li> </ul>  <p>• Дивергенция – первый анализируемый пик осциллятора ниже предыдущего, а соответствующий ему пик цены выше предыдущего.</p>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Двойная дивергенция – осциллятор сформировал три последовательных пика, каждый из которых ниже предыдущего, а цена сформировала три соответствующих им пика, каждый из которых выше предыдущего.</li> </ul>
	
Не против покупки	Значение осциллятора на анализируемом баре растет.
Не против продажи	Значение осциллятора на анализируемом баре падает.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

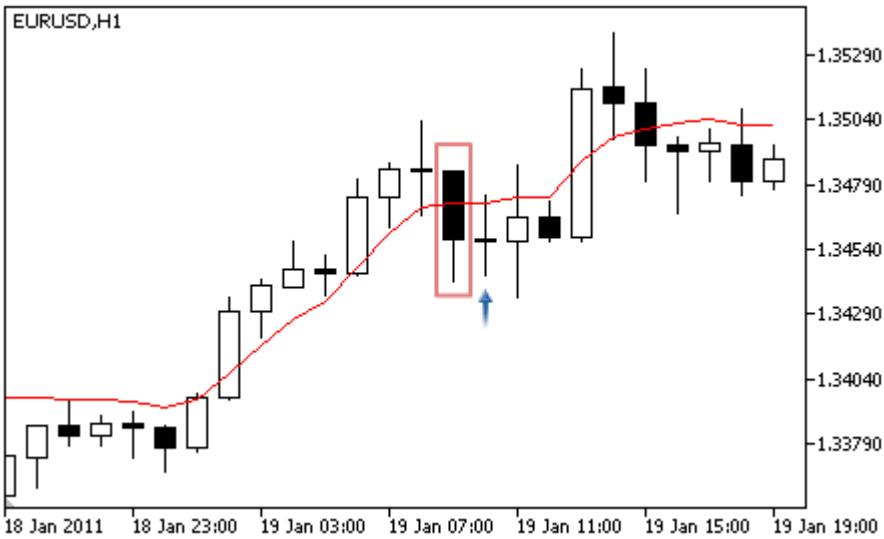
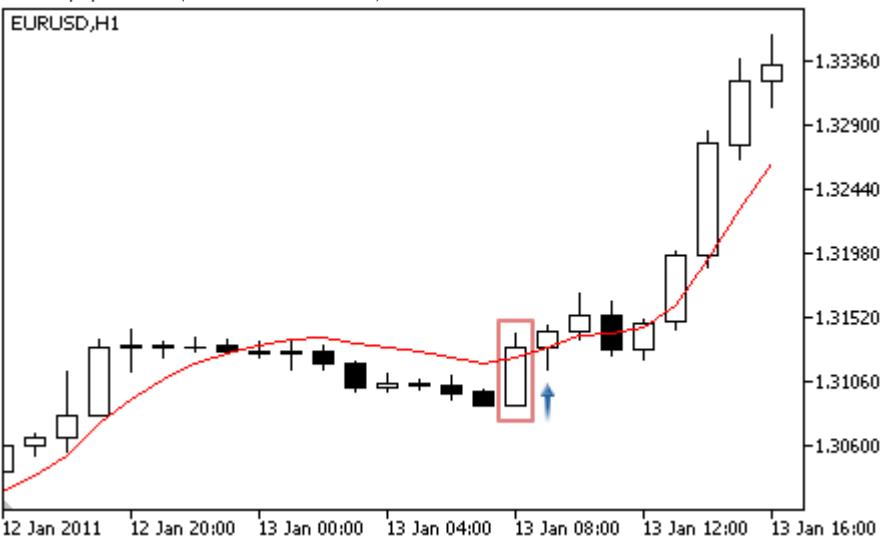
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodDeM	Период расчета осциллятора.

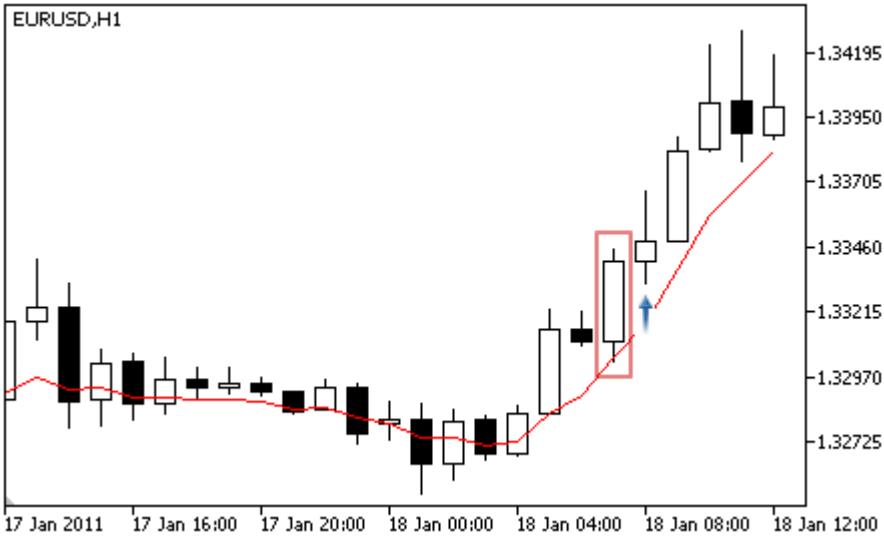
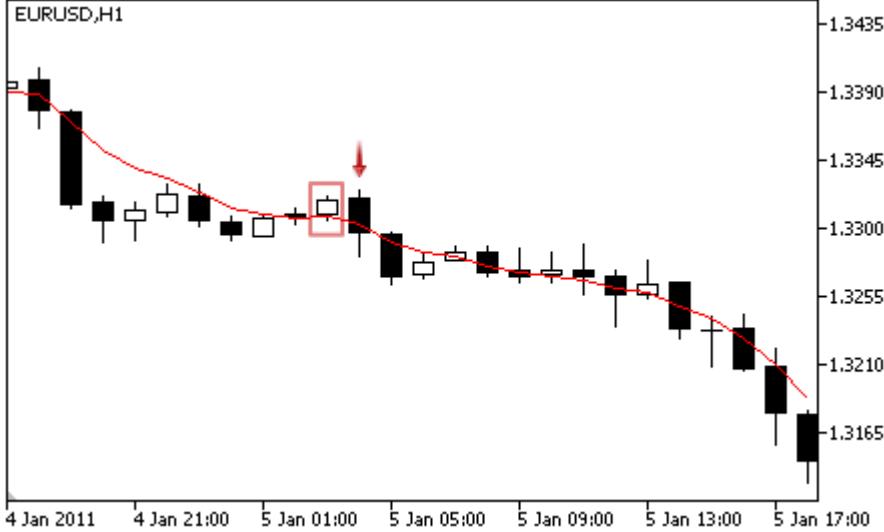
## Сигналы индикатора Double Exponential Moving Average

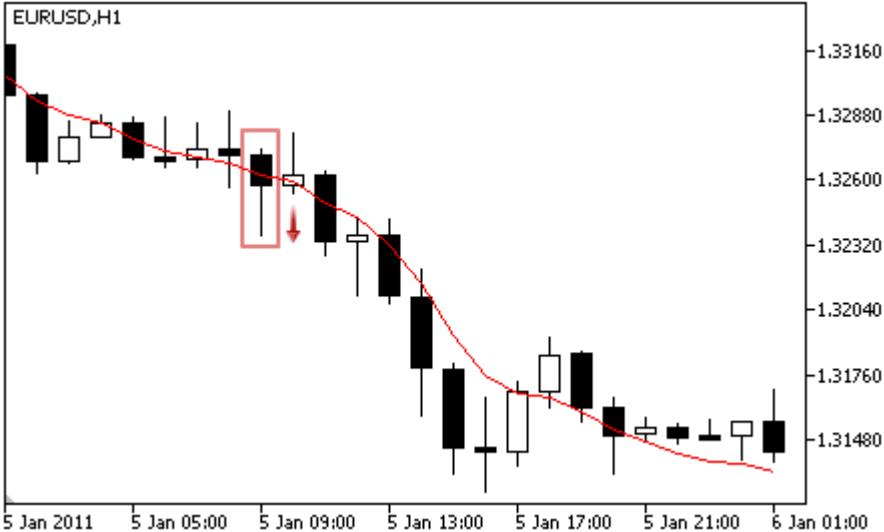
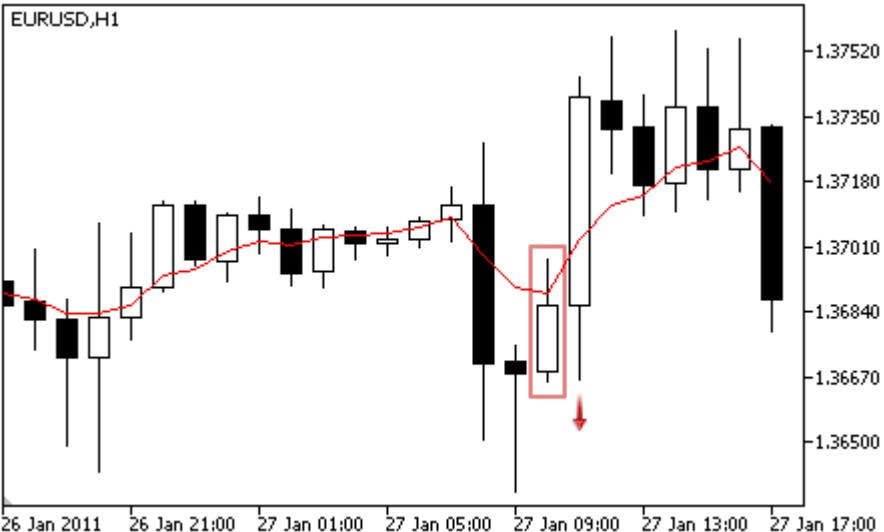
Данный модуль сигналов основан на рыночных моделях индикатора [Double Exponential Moving Average](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li><b>Несформировавшийся прокол.</b> Цена пересекла индикатор сверху вниз(цена Open анализируемого бара выше линии индикатора, а цена Close - ниже), но индикатор растет (слабый сигнал на отбой от линии индикатора).</li> </ul>  <p>EURUSD,H1 18 Jan 2011 18 Jan 23:00 19 Jan 03:00 19 Jan 07:00 19 Jan 11:00 19 Jan 15:00 19 Jan 19:00</p> <ul style="list-style-type: none"> <li><b>Пересечение скользящей средней.</b> Цена пересекла индикатор снизу вверх (цена Open анализируемого бара ниже линии индикатора, а цена Close - выше) и индикатор растет (сильный сигнал).</li> </ul>  <p>EURUSD,H1 12 Jan 2011 12 Jan 20:00 13 Jan 00:00 13 Jan 04:00 13 Jan 08:00 13 Jan 12:00 13 Jan 16:00</p>

Тип сигнала	Описание условий
	<ul style="list-style-type: none"> <li><b>Сформировавшийся прокол.</b> Цена пересекла индикатор нижней тенью (цены Open и Close анализируемого бара выше линии индикатора, а цена Low ниже) и индикатор растет (сигнал на отбой от линии индикатора).</li> </ul>  <p>The chart shows a price movement from January 17 to January 18. A red moving average line is present. A candle at approximately 18 Jan 04:00 is highlighted with a red box, indicating it crossed below the moving average's low point. A blue arrow points upwards from this candle, indicating a signal for a price rebound.</p>
За продажу	<ul style="list-style-type: none"> <li><b>Несформировавшийся прокол.</b> Цена пересекла индикатор снизу вверх (цена Open анализируемого бара ниже линии индикатора, а цена Close - выше), но индикатор падает (слабый сигнал на отбой от линии индикатора).</li> </ul>  <p>The chart shows a price movement from January 4 to January 5. A red moving average line is present. A candle at approximately 5 Jan 01:00 is highlighted with a red box, indicating it crossed above the moving average's high point. A red arrow points downwards from this candle, indicating a weak signal for a price decline.</p> <ul style="list-style-type: none"> <li><b>Пересечение скользящей средней.</b> Цена пересекла индикатор сверху вниз (цена Open анализируемого бара выше линии индикатора, а цена Close - ниже) и индикатор падает (сильный сигнал).</li> </ul>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Сформировавшийся прокол. Цена пересекла индикатор верхней тенью (цены Open и Close анализируемого бара ниже линии индикатора, а цена High выше) и индикатор падает (сигнал на отбой от линии индикатора).</li> </ul> 
Не против покупки	Цена находится выше индикатора.
Не против продажи	Цена находится ниже индикатора.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

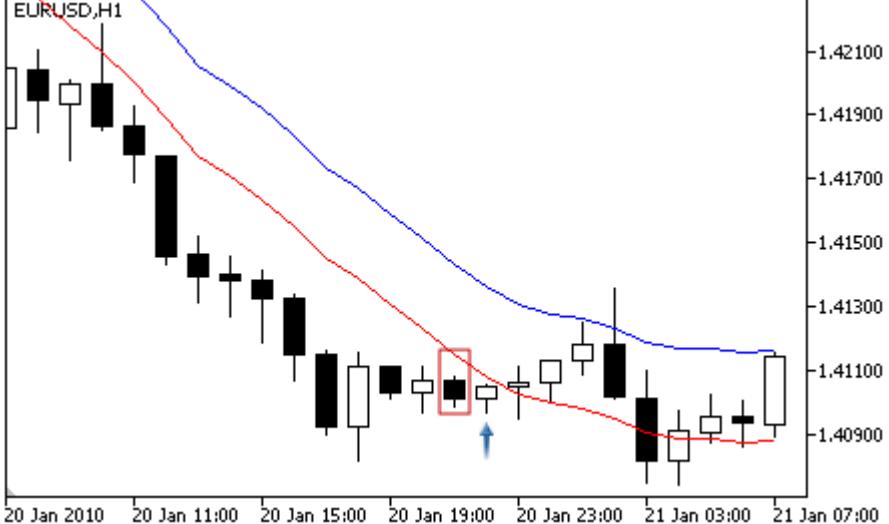
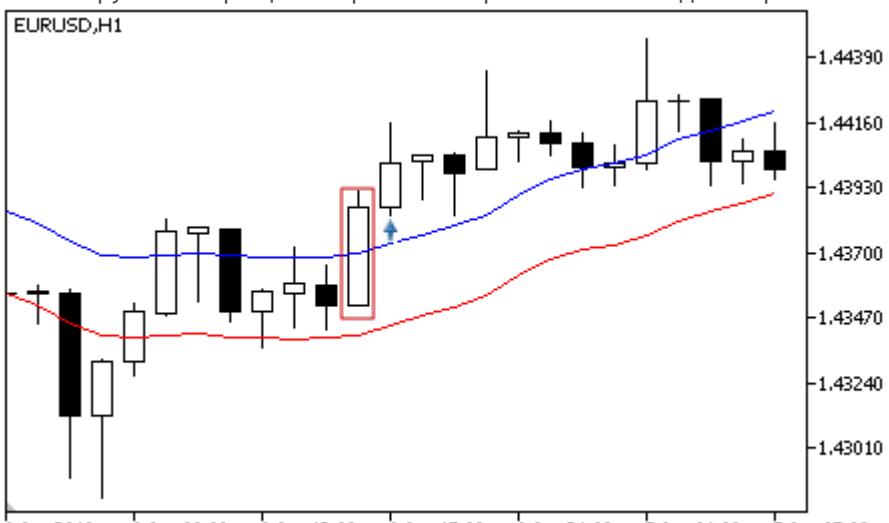
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodMA	Период усреднения индикатора.
Shift	Смещение индикатора по оси времени (в барах).
Method	<a href="#">Метод усреднения</a> .
Applied	<a href="#">Ценовой ряд</a> , на значениях которого рассчитывается индикатор.

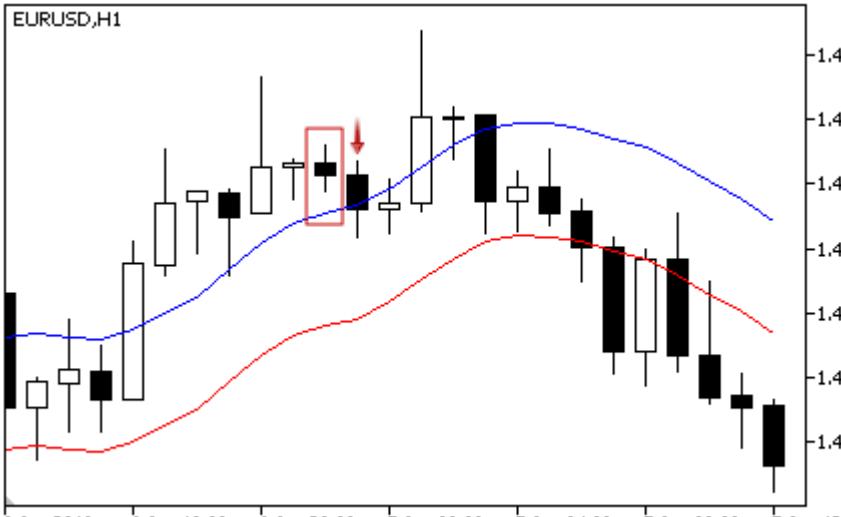
## Сигналы индикатора Envelopes

Данный модуль сигналов основан на рыночных моделях индикатора [Envelopes](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li>На анализируемом баре цена находится вблизи нижней линии индикатора.</li> </ul>  <p>The chart shows a candlestick pattern for EURUSD on a H1 timeframe from January 20, 2010, to January 21, 2010. Two parallel trend lines are drawn: a red line sloping downwards and a blue line sloping upwards. A red rectangle highlights a specific candle at 19:00 on January 20, 2010, which is positioned very close to the red lower envelope line. A blue arrow points downwards from this candle towards the red line.</p> <ul style="list-style-type: none"> <li>На анализируемом баре цена пересекла верхнюю линию индикатора.</li> </ul>  <p>The chart shows a candlestick pattern for EURUSD on a H1 timeframe from January 6, 2010, to January 7, 2010. Two parallel trend lines are shown. A red rectangle highlights a candle at 17:00 on January 6, 2010, which has just crossed above the blue upper envelope line. A blue arrow points upwards from this candle towards the blue line.</p>
За продажу	<ul style="list-style-type: none"> <li>На анализируемом баре цена находится вблизи верхней линии индикатора.</li> </ul>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>На анализируемом баре цена пересекла нижнюю линию индикатора.</li> </ul> 
Не против покупки	Сигналы отсутствуют.
Не против продажи	Сигналы отсутствуют.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

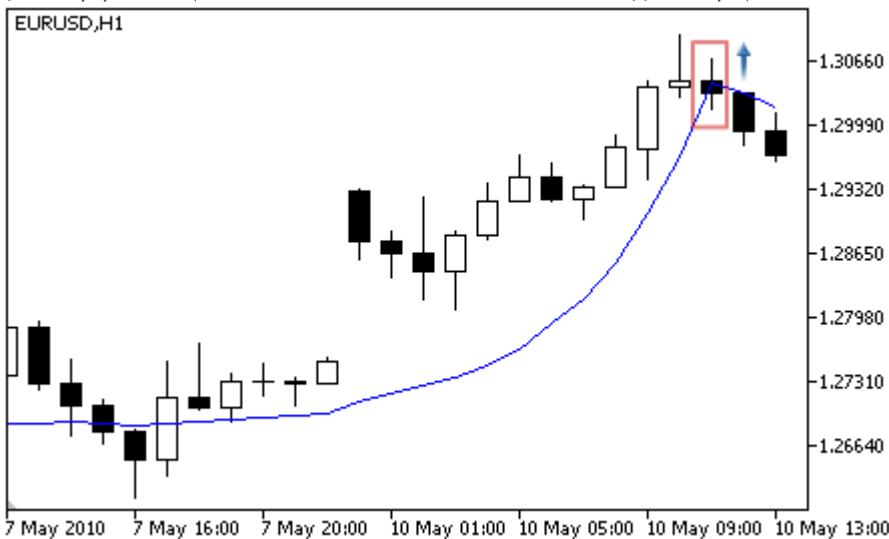
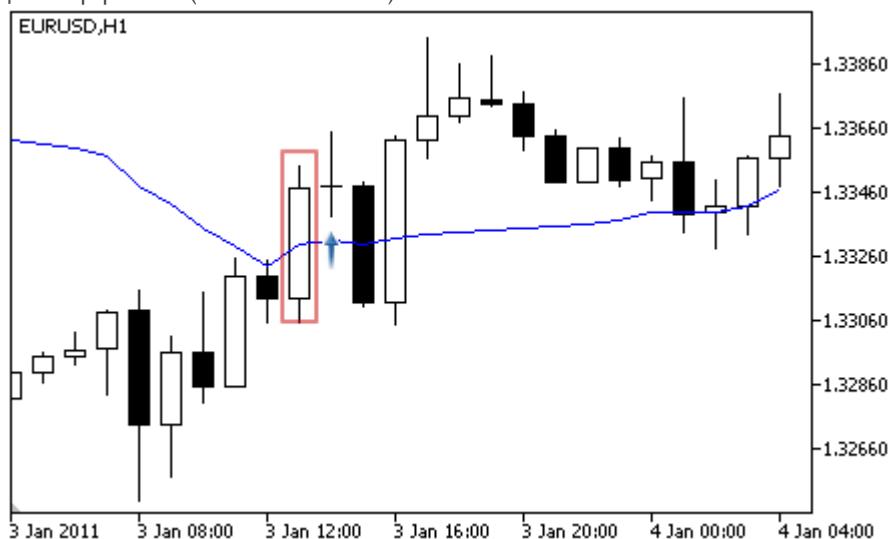
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodMA	Период расчет индикатора.
Shift	Смещение индикатора по оси времени (в барах).
Method	<a href="#">Метод усреднения.</a>
Applied	<a href="#">Ценовой ряд</a> , на значениях которого рассчитывается индикатор.
Deviation	Отклонение границ конверта от центральной линии (МА) в процентах.

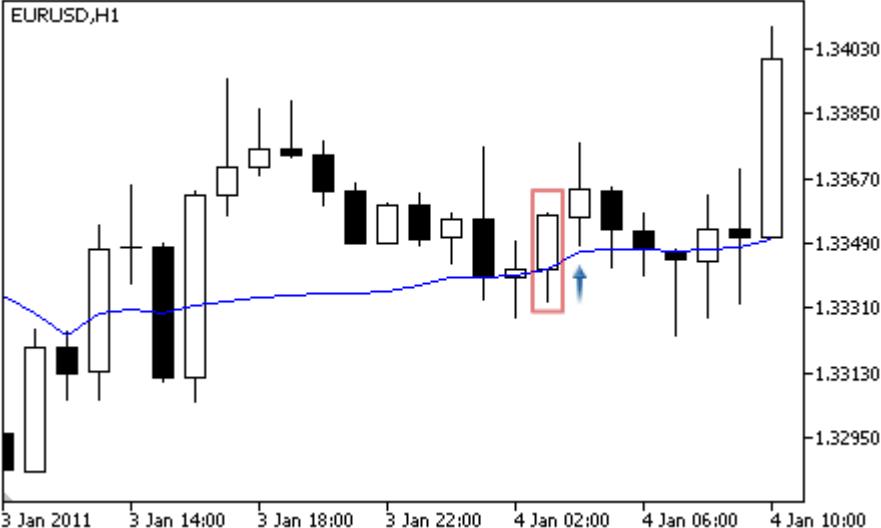
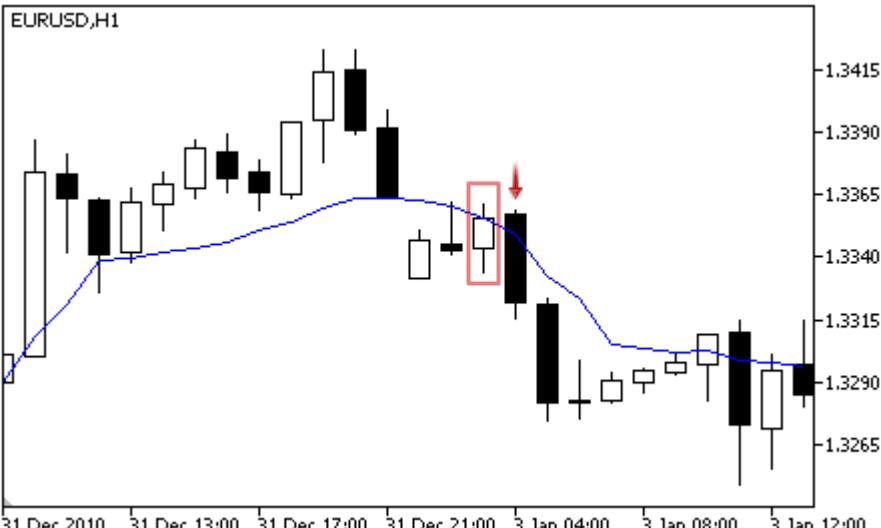
## Сигналы индикатора Fractal Adaptive Moving Average

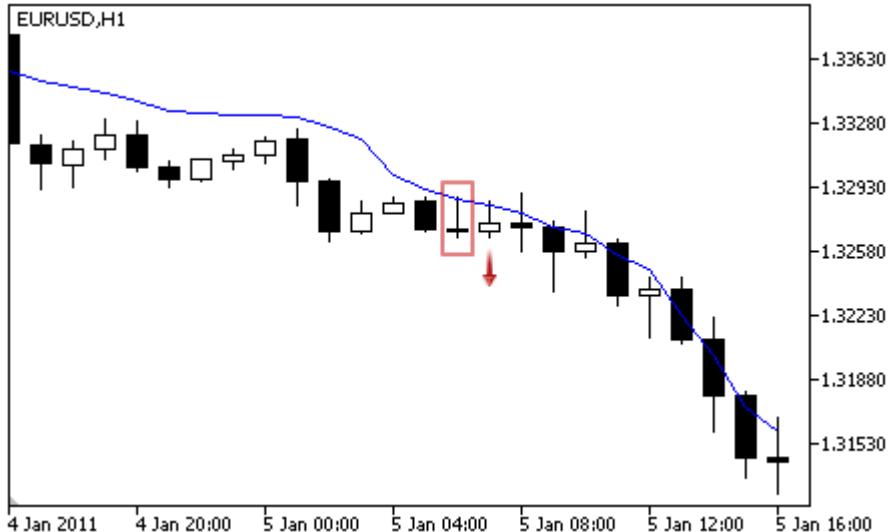
Данный модуль сигналов основан на рыночных моделях индикатора [Fractal Adaptive Moving Average](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li><b>Несформировавшийся прокол.</b> Цена пересекла индикатор сверху вниз(цена Open анализируемого бара выше линии индикатора, а цена Close - ниже), но индикатор растет (слабый сигнал на отбой от линии индикатора).</li> </ul>  <p>EURUSD,H1 7 May 2010 7 May 16:00 7 May 20:00 10 May 01:00 10 May 05:00 10 May 09:00 10 May 13:00</p> <ul style="list-style-type: none"> <li><b>Пересечение скользящей средней.</b> Цена пересекла индикатор снизу вверх (цена Open анализируемого бара ниже линии индикатора, а цена Close - выше) и индикатор растет (сильный сигнал).</li> </ul>  <p>EURUSD,H1 3 Jan 2011 3 Jan 08:00 3 Jan 12:00 3 Jan 16:00 3 Jan 20:00 4 Jan 00:00 4 Jan 04:00</p>

Тип сигнала	Описание условий
	<ul style="list-style-type: none"> <li>Сформировавшийся прокол. Цена пересекла индикатор нижней тенью (цены Open и Close анализируемого бара выше линии индикатора, а цена Low ниже) и индикатор растет (сигнал на отбой от линии индикатора).</li> </ul> 
За продажу	<ul style="list-style-type: none"> <li>Несформировавшийся прокол. Цена пересекла индикатор снизу вверх (цена Open анализируемого бара ниже линии индикатора, а цена Close - выше), но индикатор падает (слабый сигнал на отбой от линии индикатора).</li> </ul> 

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Сформировавшийся прокол. Цена пересекла индикатор верхней тенью (цены Open и Close анализируемого бара ниже линии индикатора, а цена High выше) и индикатор падает (сигнал на отбой от линии индикатора).</li> </ul> 
Не против покупки	Цена находится выше индикатора.
Не против продажи	Цена находится ниже индикатора.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodMA	Период усреднения индикатора.
Shift	Смещение индикатора по оси времени (в барах).
Method	<a href="#">Метод усреднения</a> .
Applied	<a href="#">Ценовой ряд</a> , на значениях которого рассчитывается индикатор.

## Сигналы внутридневного временного фильтра

Данный модуль сигналов основан на том, что эффективность рыночных моделей меняется во времени. Используя данный модуль, можно отфильтровывать сигналы, получаемые от других модулей, по часам и дням недели. Это позволяет улучшить качество генерируемых сигналов за счет отсечения неблагоприятных участков времени. Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	Сигналы отсутствуют.
За продажу	Сигналы отсутствуют.
Не против покупки	Текущее время и день недели удовлетворяют заданным параметрам.
Не против продажи	Текущее время и день недели удовлетворяют заданным параметрам.

### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
GoodHourOfDay	Номер единственного часа в сутках (от 0 до 23), в который будут разрешены торговые сигналы. При значении -1 сигналы разрешены круглосуточно.
BadHoursOfDay	Битовое поле, каждый бит которого соответствует часу в сутках (0 бит - 0 час, ..., 23 бит - 23 час). Если значение бита равно 0, торговые сигналы в соответствующем часе разрешены. Если значение бита равно 1, торговые сигналы в соответствующем часе запрещены. Указанное число представляется в виде двоичного и используется в виде битовой маски. Запрещенные часы имеют приоритет над разрешенными.
GoodDayOfWeek	Номер единственного дня недели (от 0 до 6, где 0 - воскресенье), в который будут

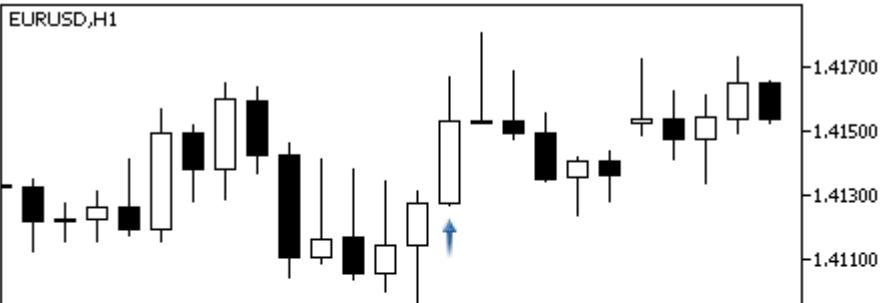
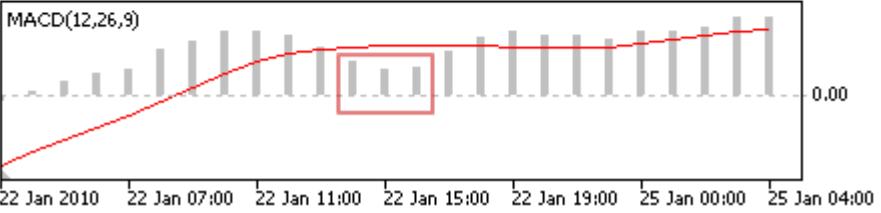
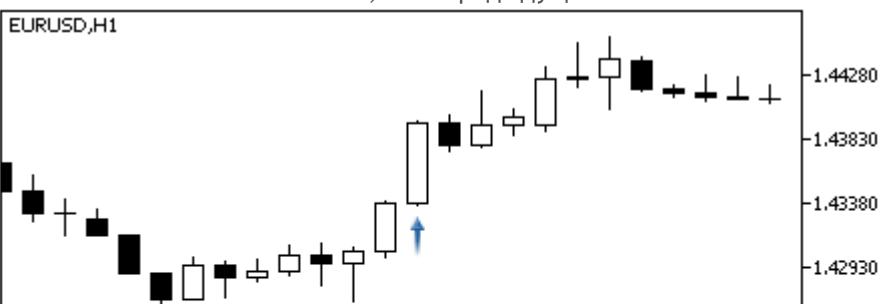
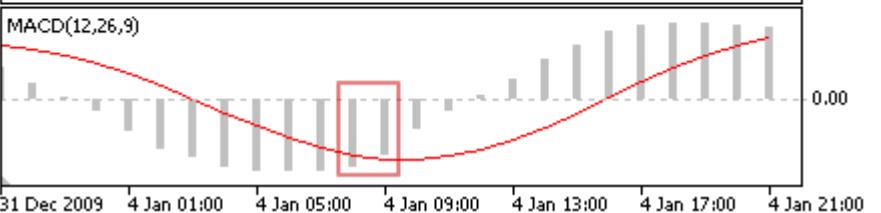
Параметр	Описание
	разрешены торговые сигналы. При значении - 1 сигналы разрешены в любой день недели.
BadDaysOfWeek	Битовое поле, каждый бит которого соответствует дню недели (0 бит - воскресенье, ..., 6 бит - суббота). Если значение бита равно 0, торговые сигналы в соответствующий день разрешены. Если значение бита равно 1, торговые сигналы в соответствующем день запрещены. Указанное число представляется в виде двоичного и используется в виде битовой маски. Запрещенные дни имеют приоритет над разрешенными.

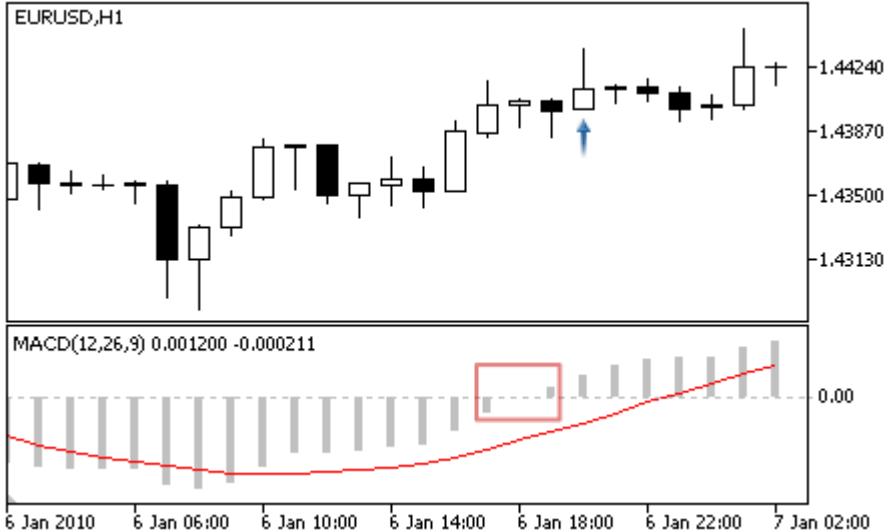
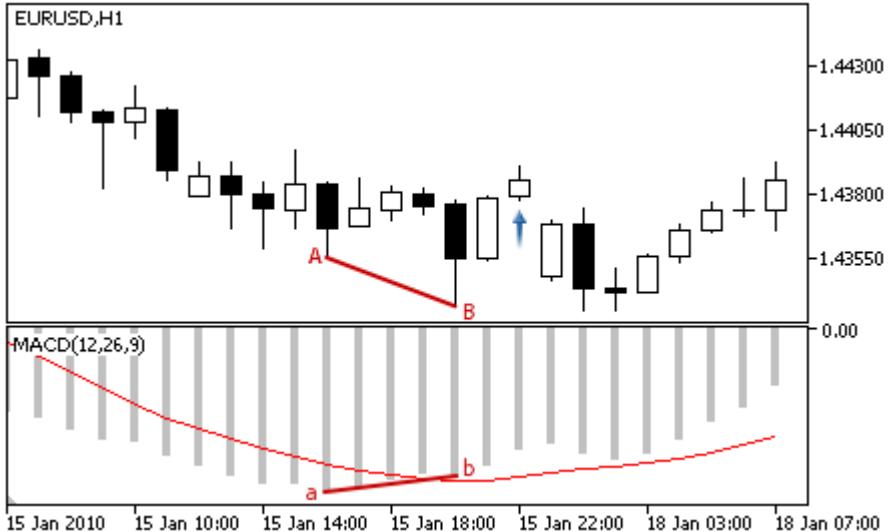
## Сигналы осциллятора MACD

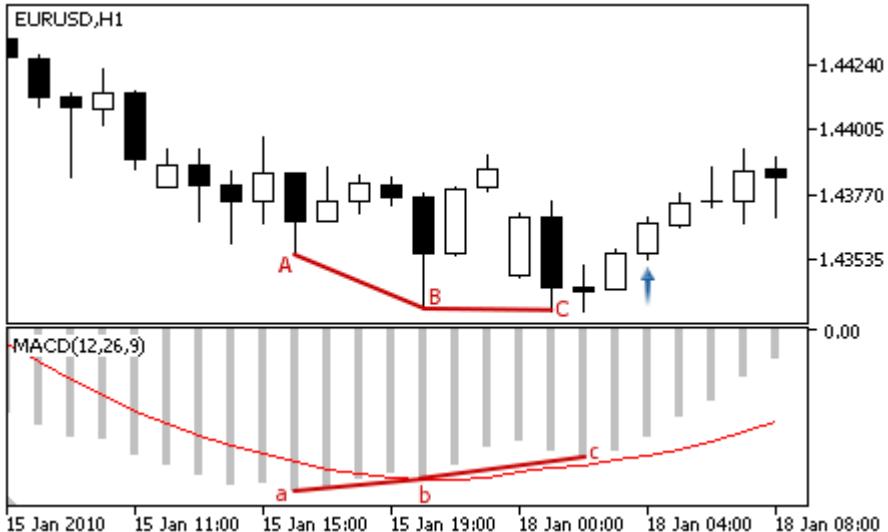
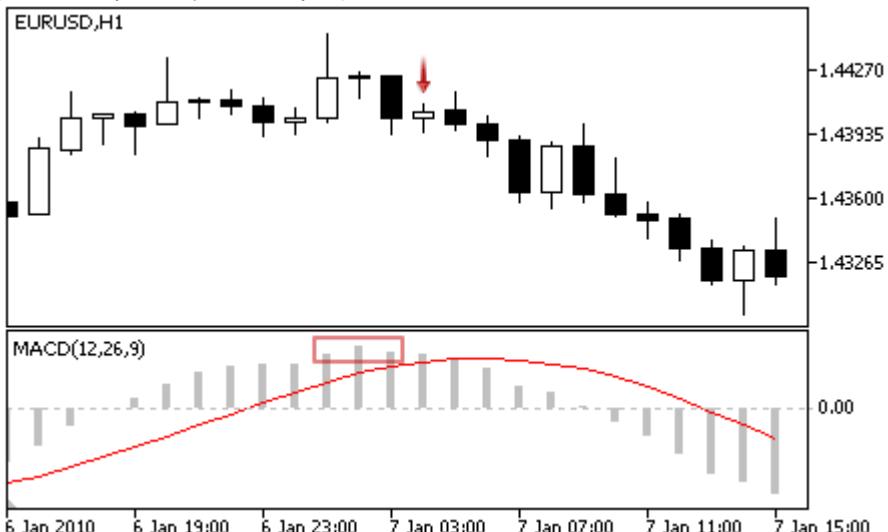
Данный модуль сигналов основан на рыночных моделях осциллятора [MACD](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

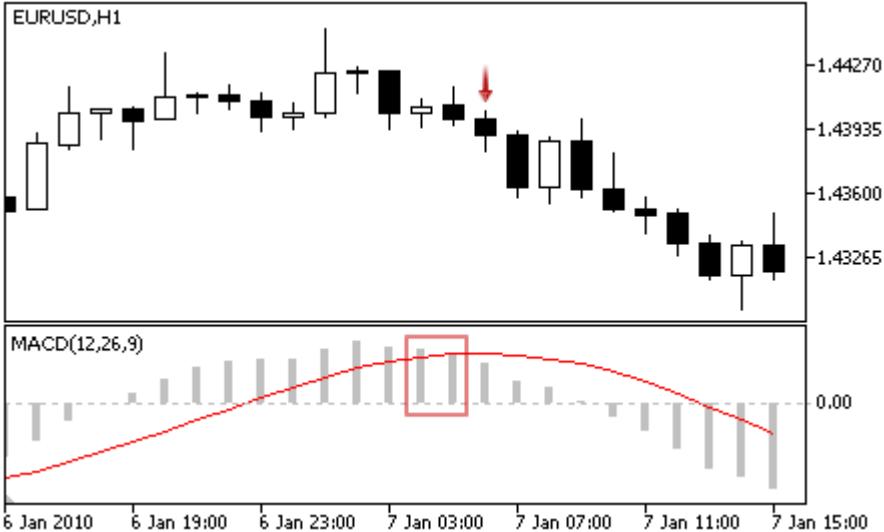
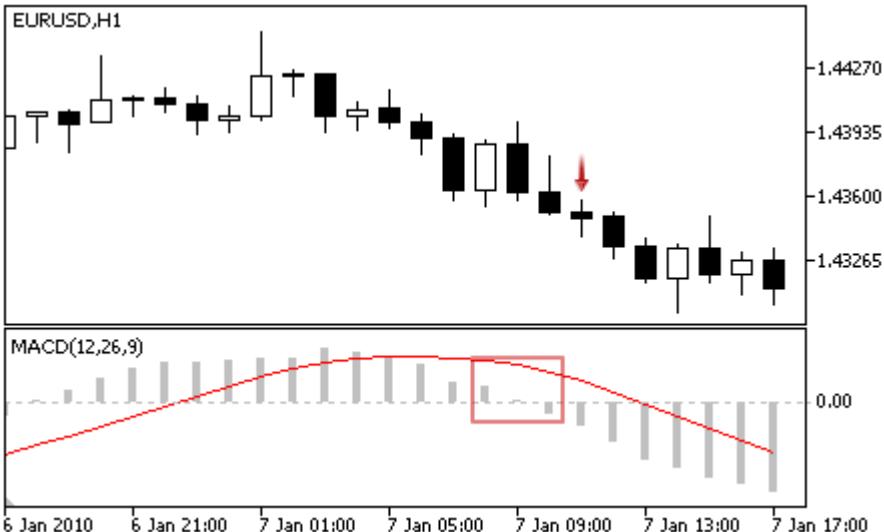
### Условия генерации сигналов

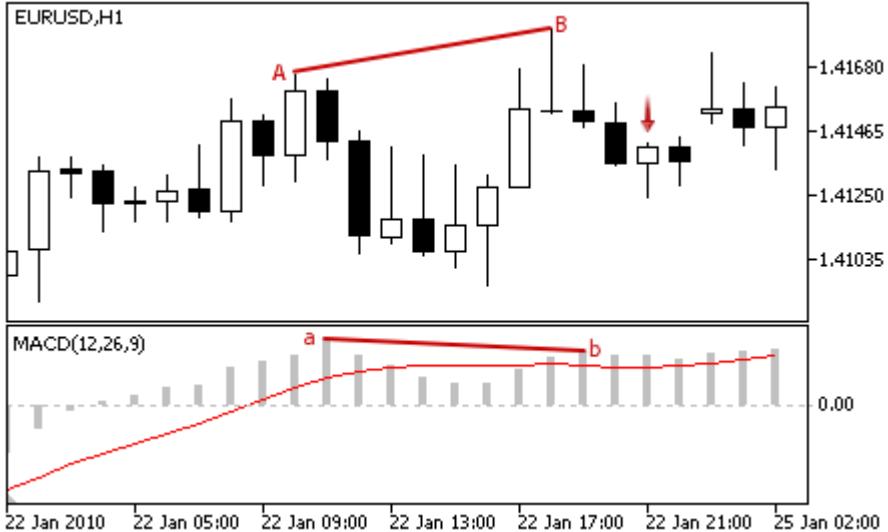
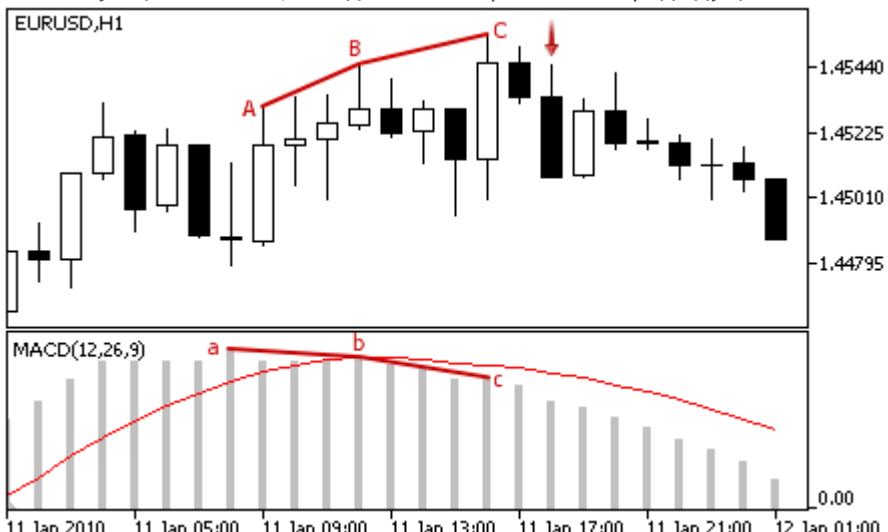
Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li>Разворот — осциллятор развернулся вверх (осциллятор растет на анализируемом баре, а на предыдущем он падал).</li> </ul>   <p>EURUSD,H1 MACD(12,26,9)</p> <p>22 Jan 2010 22 Jan 07:00 22 Jan 11:00 22 Jan 15:00 22 Jan 19:00 25 Jan 00:00 25 Jan 04:00</p> <ul style="list-style-type: none"> <li>Пересечение основной и сигнальной линии — на анализируемом баре основная линия выше сигнальной, а на предыдущем — ниже.</li> </ul>   <p>EURUSD,H1 MACD(12,26,9)</p> <p>31 Dec 2009 4 Jan 01:00 4 Jan 05:00 4 Jan 09:00 4 Jan 13:00 4 Jan 17:00 4 Jan 21:00</p> <ul style="list-style-type: none"> <li>Пересечении нулевого уровня — на анализируемом баре основная линия выше нулевого уровня, а на предыдущем — ниже.</li> </ul>

Тип сигнала	Описание условий
	 <p>EURUSD,H1</p> <p>MACD(12,26,9) 0.001200 -0.000211</p> <p>6 Jan 2010 6 Jan 06:00 6 Jan 10:00 6 Jan 14:00 6 Jan 18:00 6 Jan 22:00 7 Jan 02:00</p>
	<ul style="list-style-type: none"> <li>• <b>Дивергенция</b> — первая анализируемая впадина осциллятора мельче предыдущей, а соответствующая ей впадина цены глубже предыдущей.</li> </ul>  <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>15 Jan 2010 15 Jan 10:00 15 Jan 14:00 15 Jan 18:00 15 Jan 22:00 18 Jan 03:00 18 Jan 07:00</p>

Тип сигнала	Описание условий
	 <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>15 Jan 2010 15 Jan 11:00 15 Jan 15:00 15 Jan 19:00 18 Jan 00:00 18 Jan 04:00 18 Jan 08:00</p>
За продажу	<ul style="list-style-type: none"> <li>Разворот — осциллятор развернулся вниз (осциллятор падает на анализируемом баре, а на предыдущем он рос).</li> </ul>  <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>6 Jan 2010 6 Jan 19:00 6 Jan 23:00 7 Jan 03:00 7 Jan 07:00 7 Jan 11:00 7 Jan 15:00</p> <ul style="list-style-type: none"> <li>Пересечение основной и сигнальной линии — на анализируемом баре основная линия ниже сигнальной, а на предыдущем — выше.</li> </ul>

Тип сигнала	Описание условий
	 <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>6 Jan 2010 6 Jan 19:00 6 Jan 23:00 7 Jan 03:00 7 Jan 07:00 7 Jan 11:00 7 Jan 15:00</p> <ul style="list-style-type: none"> <li>Пересечении нулевого уровня — на анализируемом баре основная линия ниже нулевого уровня, а на предыдущем — выше.</li> </ul>
	 <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>6 Jan 2010 6 Jan 21:00 7 Jan 01:00 7 Jan 05:00 7 Jan 09:00 7 Jan 13:00 7 Jan 17:00</p> <ul style="list-style-type: none"> <li>Дивергенция — первый анализируемый пик осциллятора ниже предыдущего, а соответствующий ему пик цены выше предыдущего.</li> </ul>

Тип сигнала	Описание условий
	 <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>22 Jan 2010 22 Jan 05:00 22 Jan 09:00 22 Jan 13:00 22 Jan 17:00 22 Jan 21:00 25 Jan 02:00</p> <ul style="list-style-type: none"> <li>Двойная дивергенция – осциллятор сформировал три последовательных пика, каждый из которых ниже предыдущего, а цена сформировала три соответствующих им пика, каждый из которых выше предыдущего.</li> </ul>  <p>EURUSD,H1</p> <p>MACD(12,26,9)</p> <p>11 Jan 2010 11 Jan 05:00 11 Jan 09:00 11 Jan 13:00 11 Jan 17:00 11 Jan 21:00 12 Jan 01:00</p>
Не против покупки	Значение осциллятора на анализируемом баре растет.
Не против продажи	Значение осциллятора на анализируемом баре падает.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

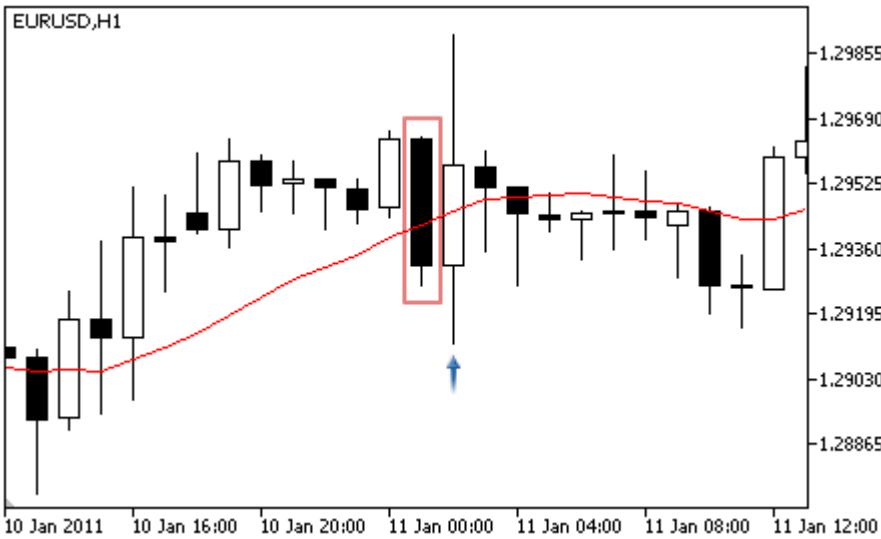
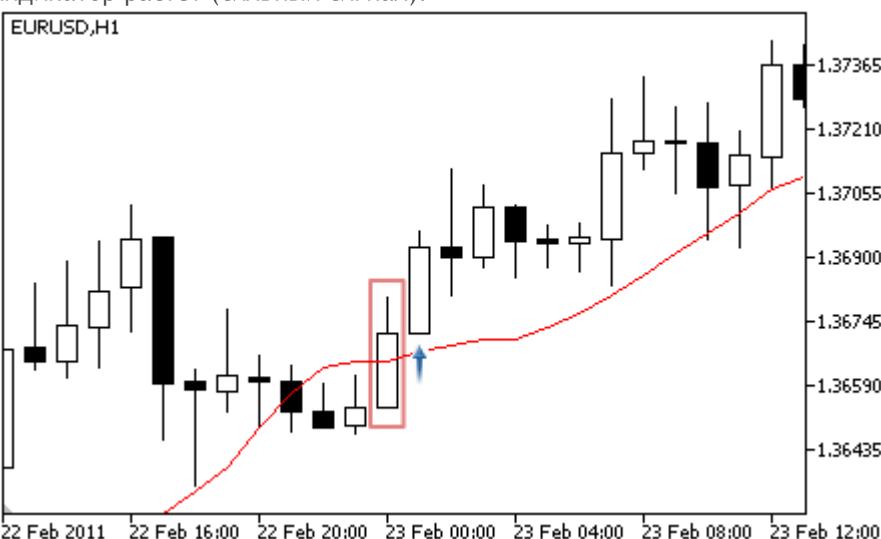
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodFast	Период расчета быстрой ЕМА.
PeriodSlow	Период расчета медленной ЕМА.
PeriodSignal	Период сглаживания.
Applied	<a href="#">Ценовой ряд</a> , на значениях которого рассчитывается осциллятор.

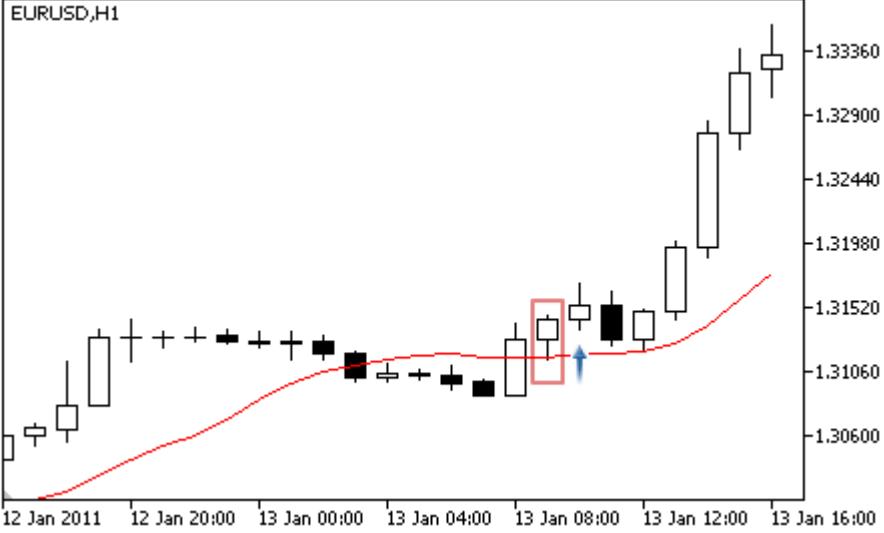
## Сигналы индикатора Moving Average

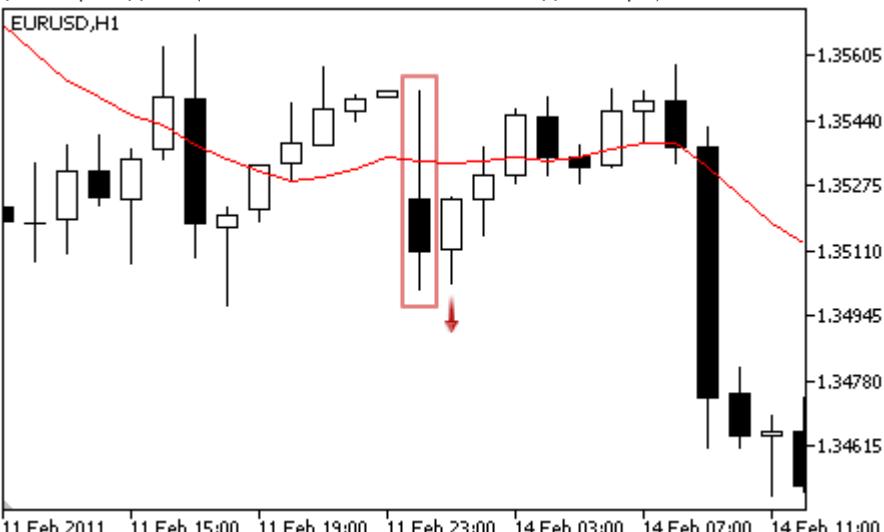
Данный модуль сигналов основан на рыночных моделях индикатора [Moving Average](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li><b>Несформировавшийся прокол.</b> Цена пересекла индикатор сверху вниз(цена Open анализируемого бара выше линии индикатора, а цена Close - ниже), но индикатор растет (слабый сигнал на отбой от линии индикатора).</li> </ul> 
	<ul style="list-style-type: none"> <li><b>Пересечение скользящей средней.</b> Цена пересекла индикатор снизу вверх (цена Open анализируемого бара ниже линии индикатора, а цена Close - выше) и индикатор растет (сильный сигнал).</li> </ul> 

Тип сигнала	Описание условий
	<ul style="list-style-type: none"> <li>Сформировавшийся прокол. Цена пересекла индикатор нижней тенью (цены Open и Close анализируемого бара выше линии индикатора, а цена Low ниже) и индикатор растет (сигнал на отбой от линии индикатора).</li> </ul> 
За продажу	<ul style="list-style-type: none"> <li>Несформировавшийся прокол. Цена пересекла индикатор снизу вверх (цена Open анализируемого бара ниже линии индикатора, а цена Close - выше), но индикатор падает (слабый сигнал на отбой от линии индикатора).</li> <li>Пересечение скользящей средней. Цена пересекла индикатор сверху вниз (цена Open анализируемого бара выше линии индикатора, а цена Close - ниже) и индикатор падает (сильный сигнал).</li> </ul> 

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Сформировавшийся прокол. Цена пересекла индикатор верхней тенью (цены Open и Close анализируемого бара ниже линии индикатора, а цена High выше) и индикатор падает (сигнал на отбой от линии индикатора).</li> </ul> 
Не против покупки	Цена находится выше индикатора.
Не против продажи	Цена находится ниже индикатора.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

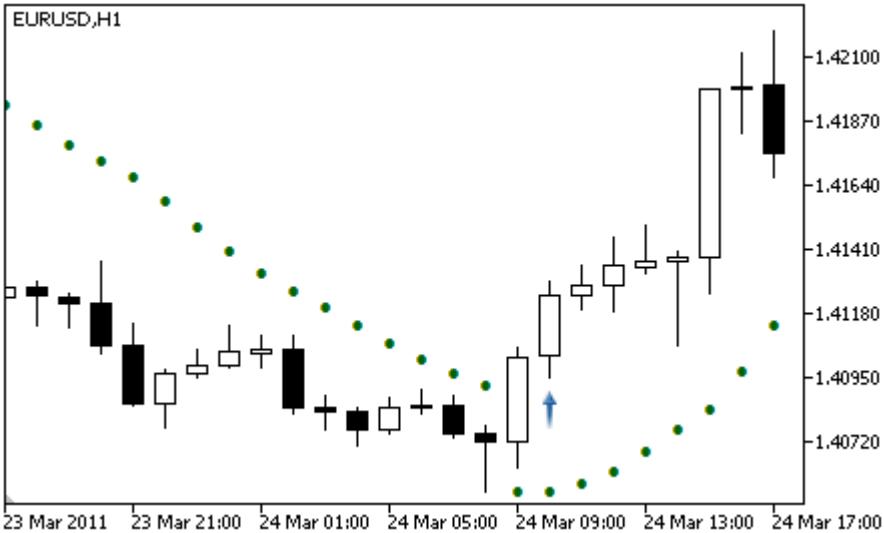
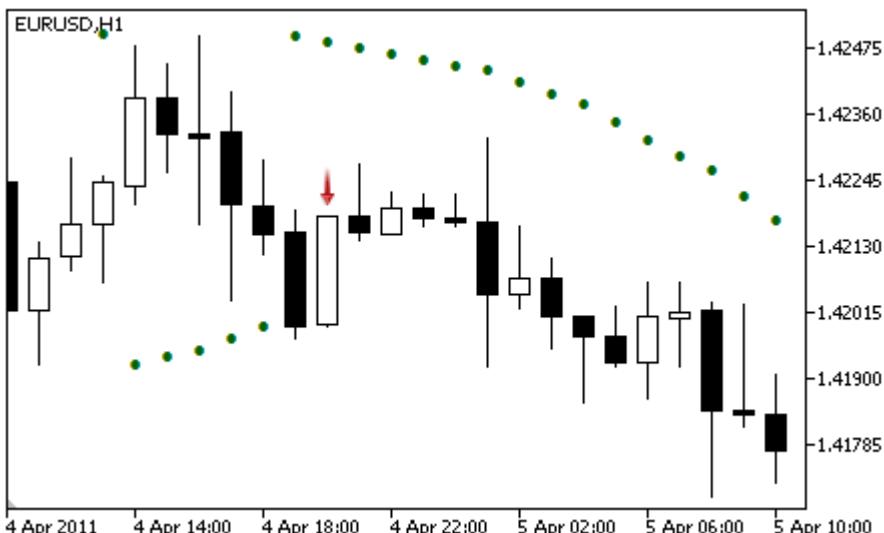
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodMA	Период усреднения индикатора.
Shift	Смещение индикатора по оси времени (в барах).
Method	<a href="#">Метод усреднения</a> .
Applied	<a href="#">Ценовой ряд</a> , на значениях которого рассчитывается индикатор.

## Сигналы индикатора Parabolic SAR

Данный модуль сигналов основан на рыночных моделях индикатора [Parabolic SAR](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<p><b>Разворот</b> – на анализируемом баре индикатор ниже цены, а на предыдущем – выше.</p> 
За продажу	<p><b>Разворот</b> – на анализируемом баре индикатор выше цены, а на предыдущем – ниже.</p> 
Не против покупки	Цена находится выше индикатора.

Тип сигнала	Описание условий
Не против продажи	Цена находится ниже индикатора.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

## Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

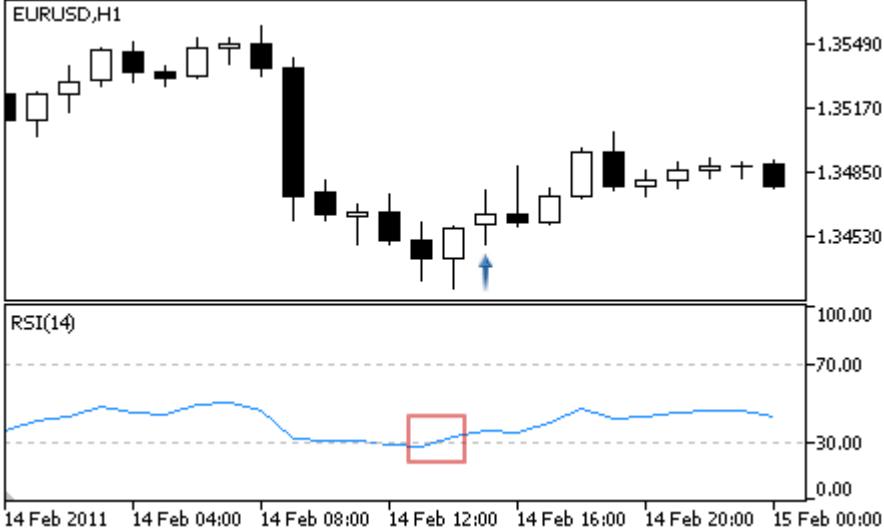
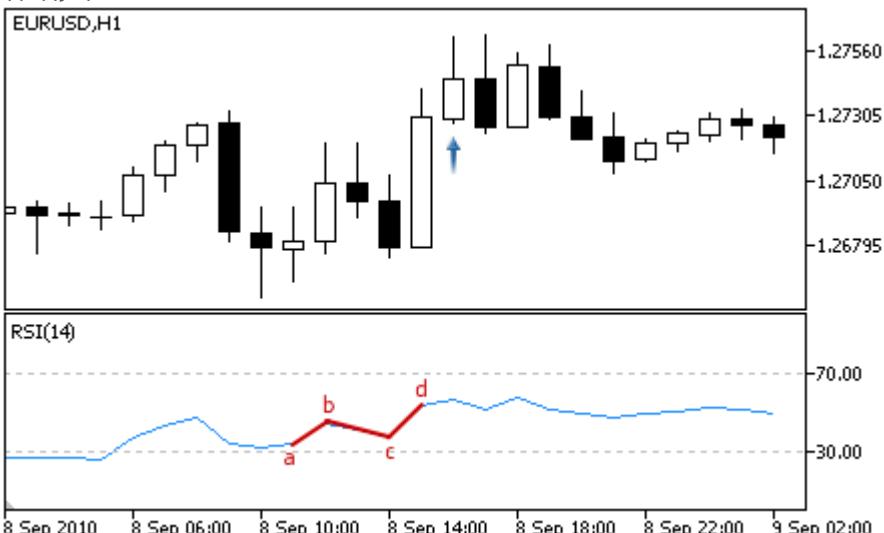
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
Step	Шаг изменения цены.
Maximum	Максимальный коэффициент скорости сближения индикатора с ценой.

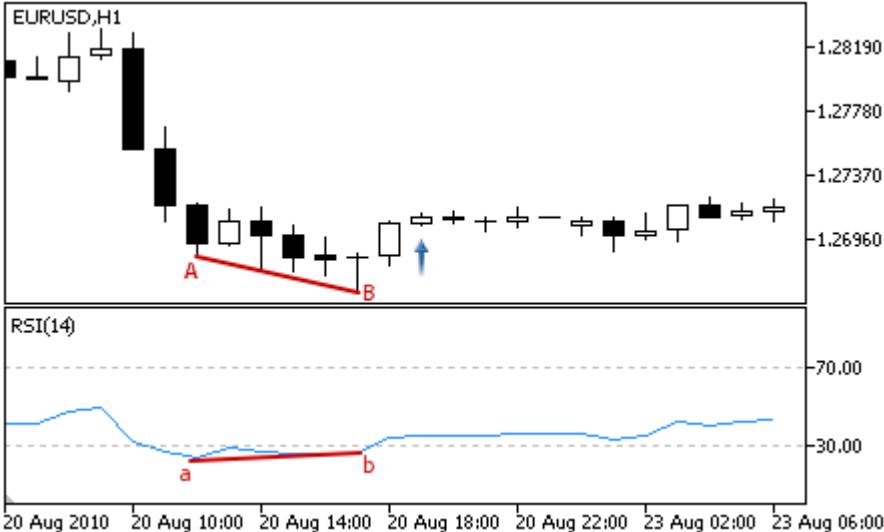
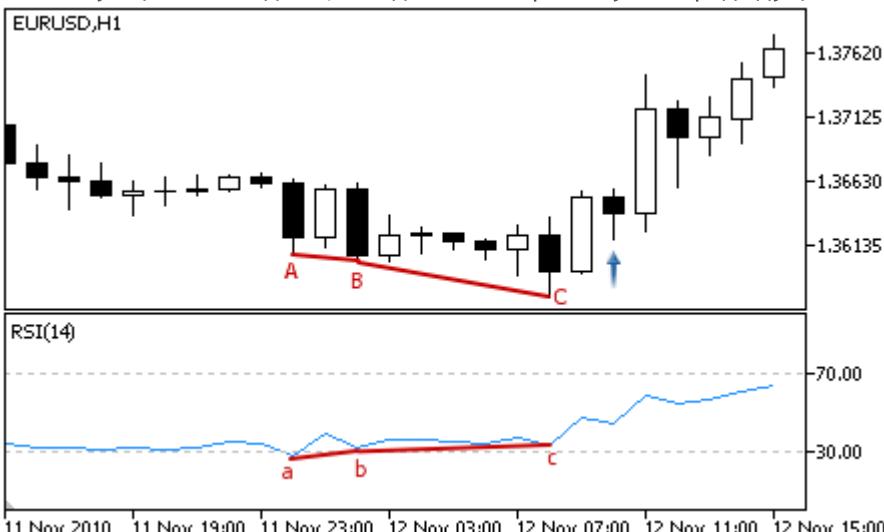
## Сигналы осциллятора Relative Strength Index

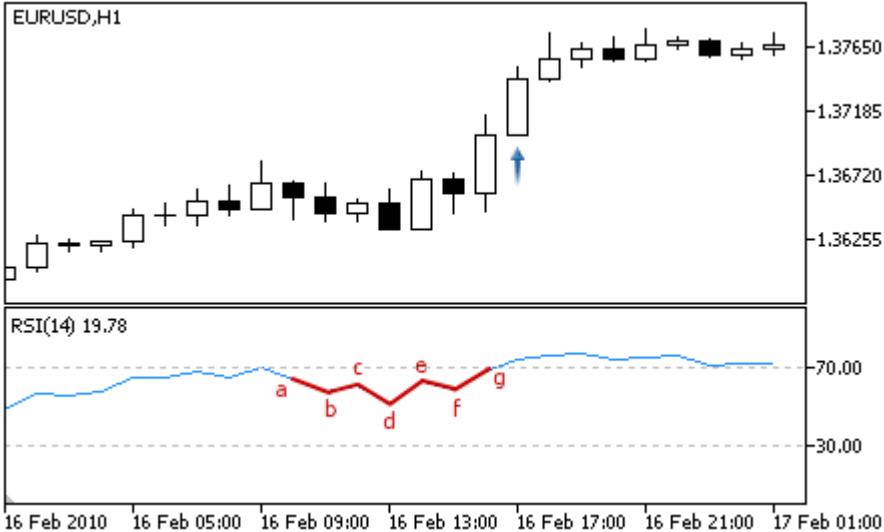
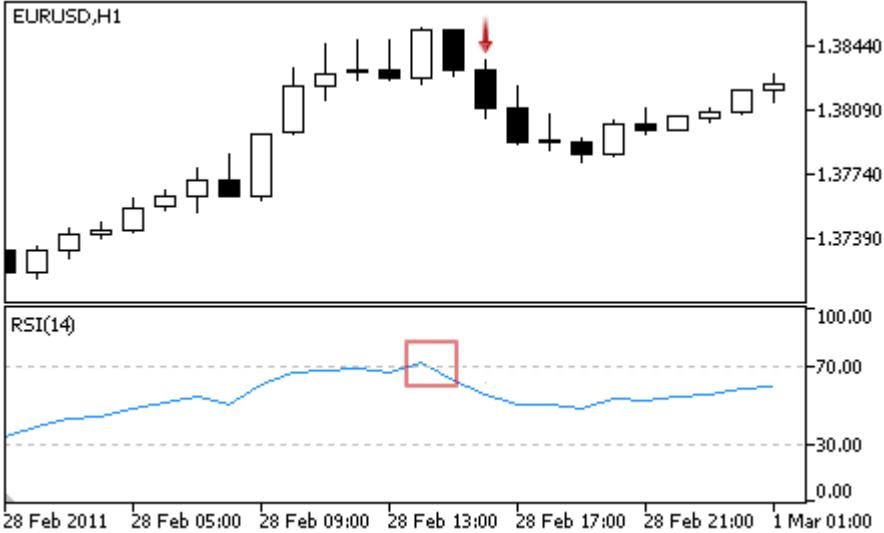
Данный модуль сигналов основан на рыночных моделях осциллятора [Relative Strength Index](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

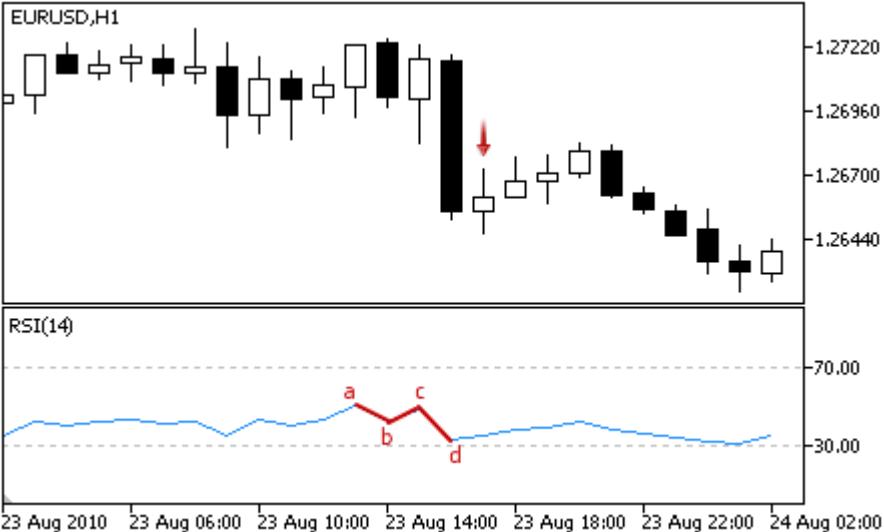
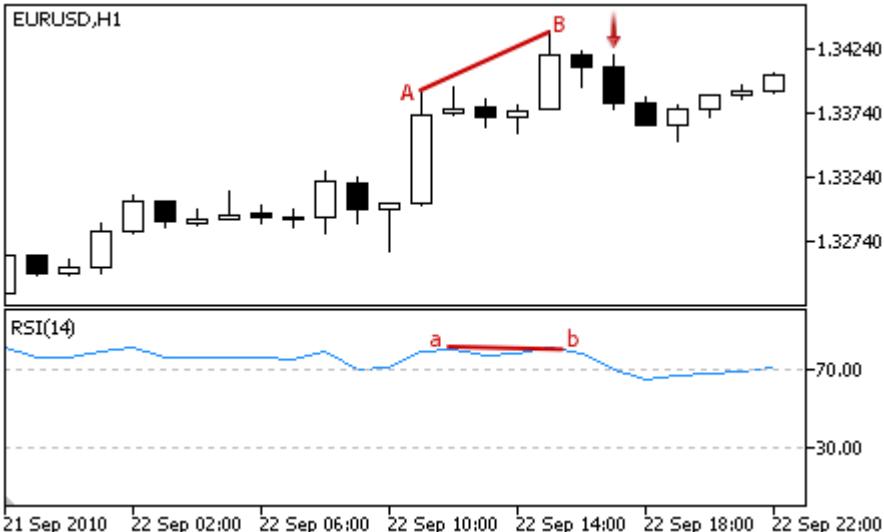
### Условия генерации сигналов

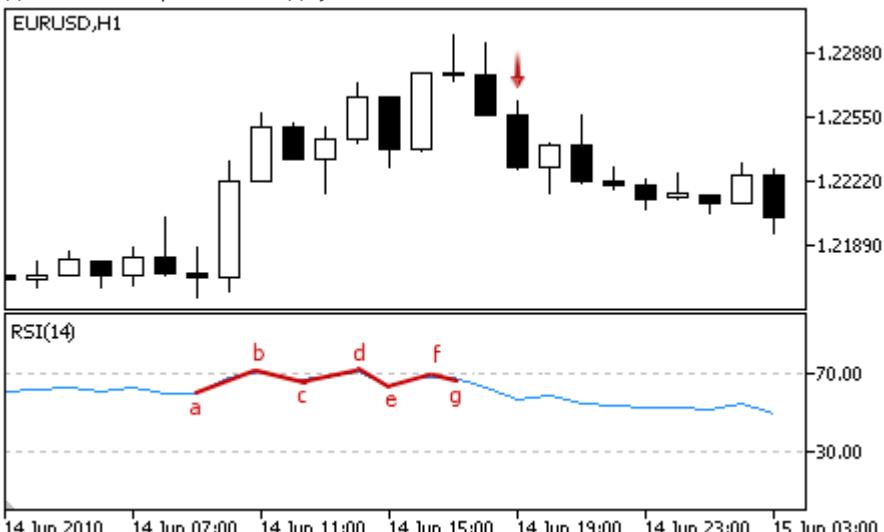
Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li><b>Разворот за уровнем перепроданности</b> – осциллятор развернулся вверх и его значение на анализируемом баре находится за уровнем перепроданности (по умолчанию 30).</li>  <p>The chart shows EURUSD price action and the RSI(14) indicator. The RSI line crosses above the 30 level, indicating a potential buy signal. A red box highlights the RSI value around 30, and a blue arrow points upwards from the candlestick chart.</p> <li><b>Неудавшийся размах</b> – на анализируемом баре осциллятор поднялся выше предыдущего пика.</li>  <p>The chart shows EURUSD price action and the RSI(14) indicator. The RSI line rises above its previous peak, labeled 'd', but then falls back below it, labeled 'c'. Points 'a' and 'b' mark local peaks before 'd'. A blue arrow points upwards from the candlestick chart.</p> <li><b>Дивергенция</b> – первая анализируемая впадина осциллятора мельче предыдущей, а соответствующая ей впадина цены глубже предыдущей.</li> </ul>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Двойная дивергенция – осциллятор сформировал три последовательных впадины, каждая из которых мельче предыдущей, а цена сформировала три соответствующие им впадины, каждая из которых глубже предыдущей.</li> </ul>  <ul style="list-style-type: none"> <li>Голова/Плечи – осциллятор сформировал три последовательных впадины, средняя из которых глубже двух остальных.</li> </ul>

Тип сигнала	Описание условий
	 <p>EURUSD,H1</p> <p>RSI(14) 19.78</p> <p>16 Feb 2010 16 Feb 05:00 16 Feb 09:00 16 Feb 13:00 16 Feb 17:00 16 Feb 21:00 17 Feb 01:00</p>
За продажу	<ul style="list-style-type: none"> <li>Разворот за уровнем перекупленности — осциллятор развернулся вниз и его значение на анализируемом баре находится за уровнем перекупленности (по умолчанию 70).</li> </ul>  <p>EURUSD,H1</p> <p>RSI(14)</p> <p>28 Feb 2011 28 Feb 05:00 28 Feb 09:00 28 Feb 13:00 28 Feb 17:00 28 Feb 21:00 1 Mar 01:00</p> <ul style="list-style-type: none"> <li>Неудавшийся размах — на анализируемом баре осциллятор опустился ниже предыдущей впадины.</li> </ul>

Тип сигнала	Описание условий
	 <p>EURUSD,H1</p> <p>RSI(14)</p> <p>23 Aug 2010 23 Aug 06:00 23 Aug 10:00 23 Aug 14:00 23 Aug 18:00 23 Aug 22:00 24 Aug 02:00</p>
	<ul style="list-style-type: none"> <li>• Дивергенция – первый анализируемый пик осциллятора ниже предыдущего, а соответствующий ему пик цены выше предыдущего.</li> </ul>  <p>EURUSD,H1</p> <p>RSI(14)</p> <p>21 Sep 2010 22 Sep 02:00 22 Sep 06:00 22 Sep 10:00 22 Sep 14:00 22 Sep 18:00 22 Sep 22:00</p>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Голова/Плечи – осциллятор сформировал три последовательных пика, средний из которых выше двух остальных.</li> </ul> 
Не против покупки	Значение осциллятора на анализируемом баре растет.
Не против продажи	Значение осциллятора на анализируемом баре падает.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

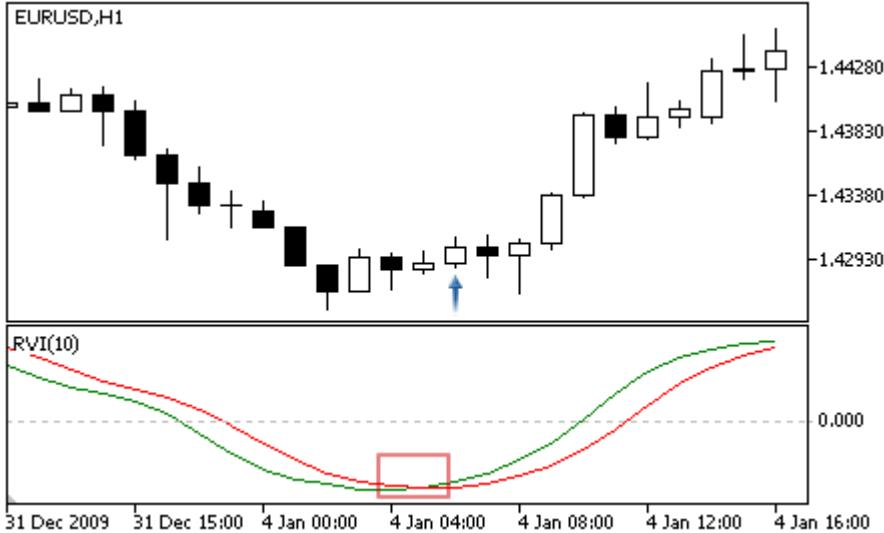
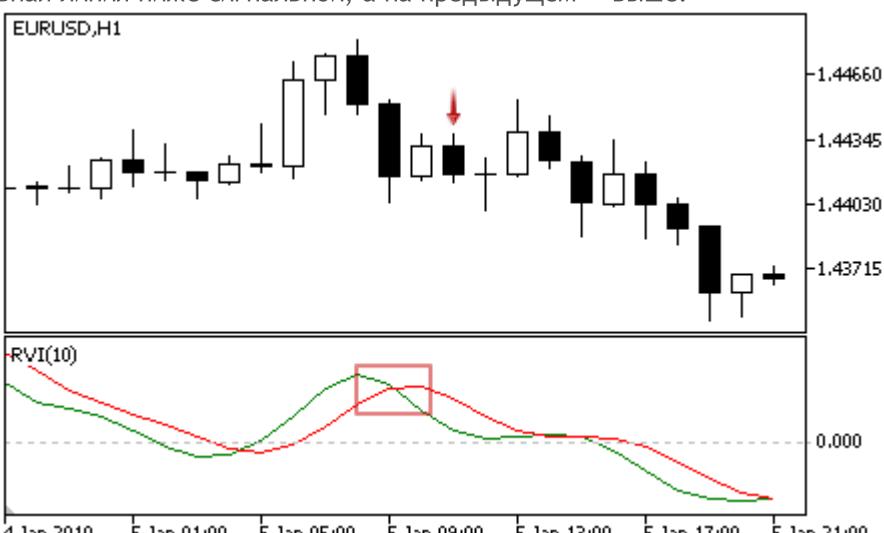
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodRSI	Период расчета осциллятора.
Applied	<a href="#">Ценовой ряд</a> , на значениях которого рассчитывается осциллятор.

## Сигналы осциллятора Relative Vigor Index

Данный модуль сигналов основан на рыночных моделях осциллятора [Relative Vigor Index](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<p>Пересечение основной и сигнальной линии – на анализируемом баре основная линия выше сигнальной, а на предыдущем – ниже.</p> 
За продажу	<p>Пересечение основной и сигнальной линии – на анализируемом баре основная линия ниже сигнальной, а на предыдущем – выше.</p> 
Не против покупки	Значение осциллятора на анализируемом баре растет.

Тип сигнала	Описание условий
Не против продажи	Значение осциллятора на анализируемом баре падает.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

## Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

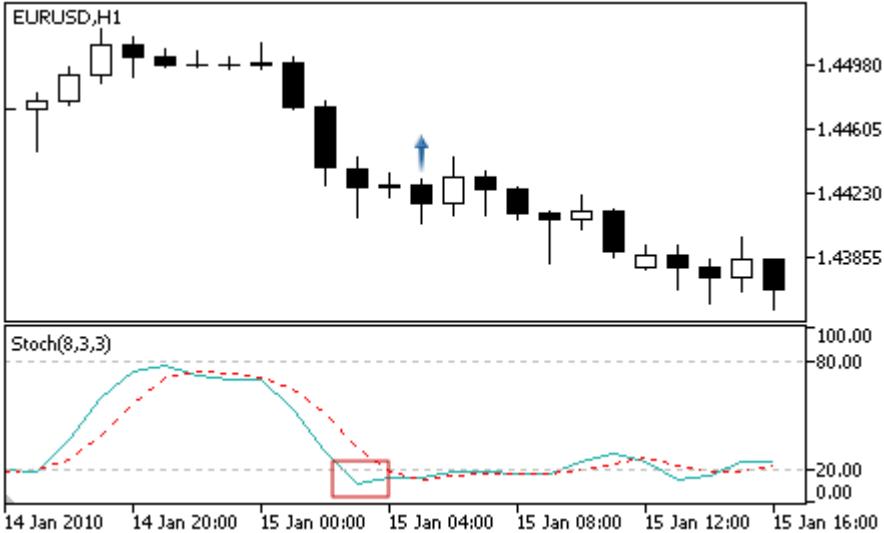
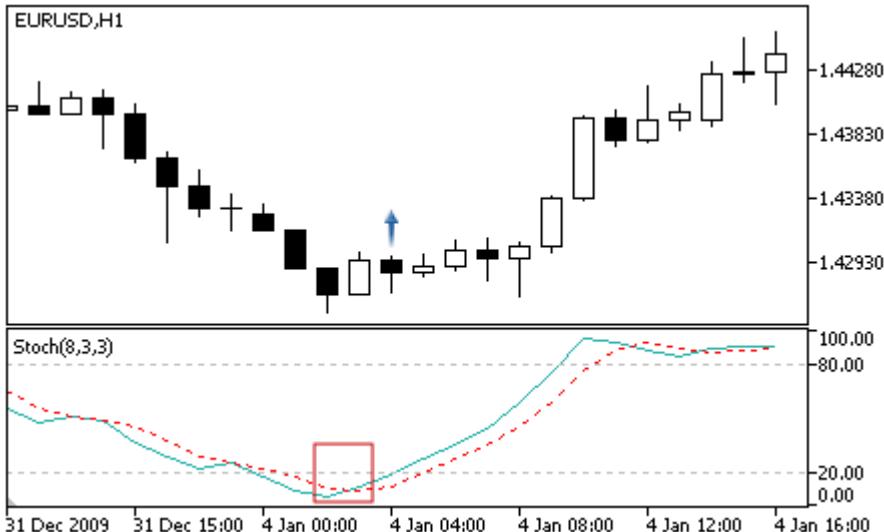
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodRVI	Период расчета осциллятора.

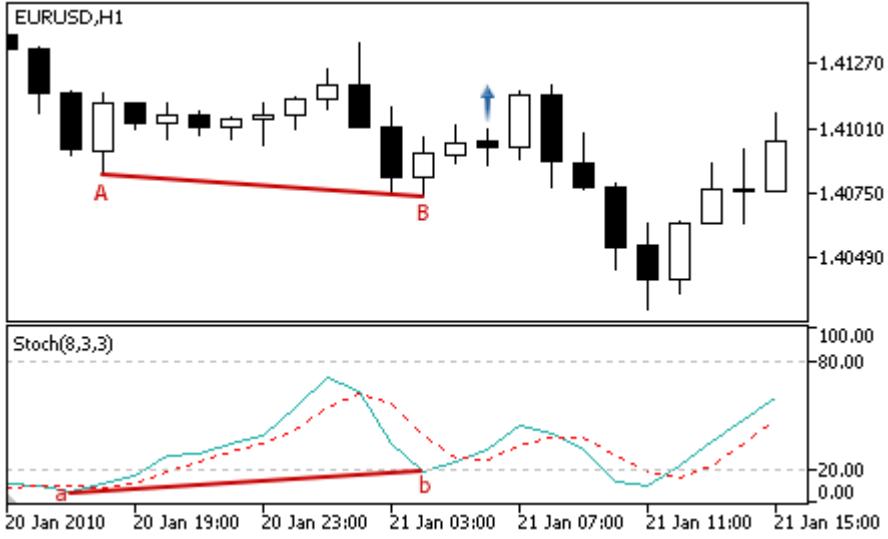
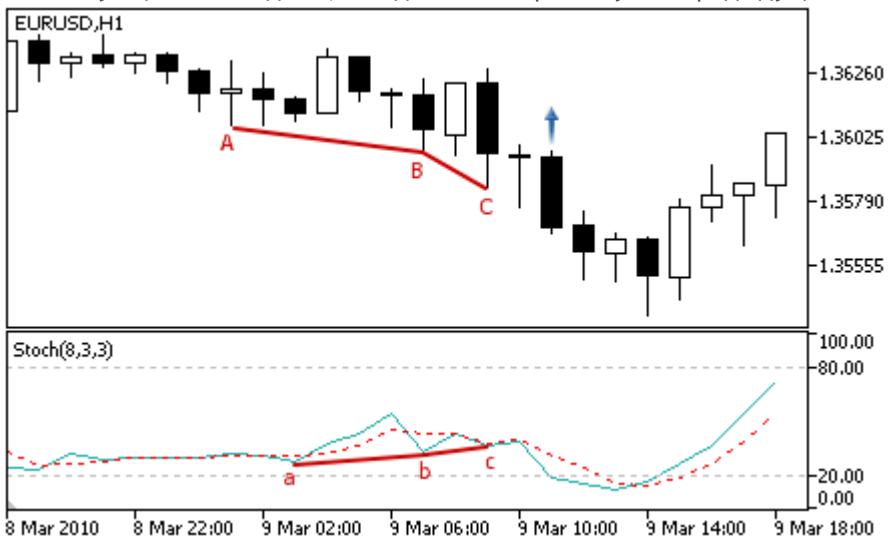
## Сигналы осциллятора Stochastic

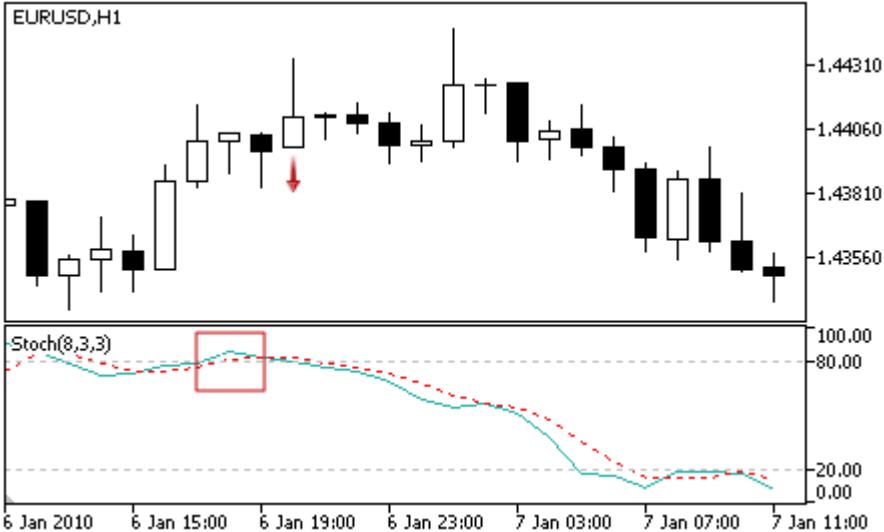
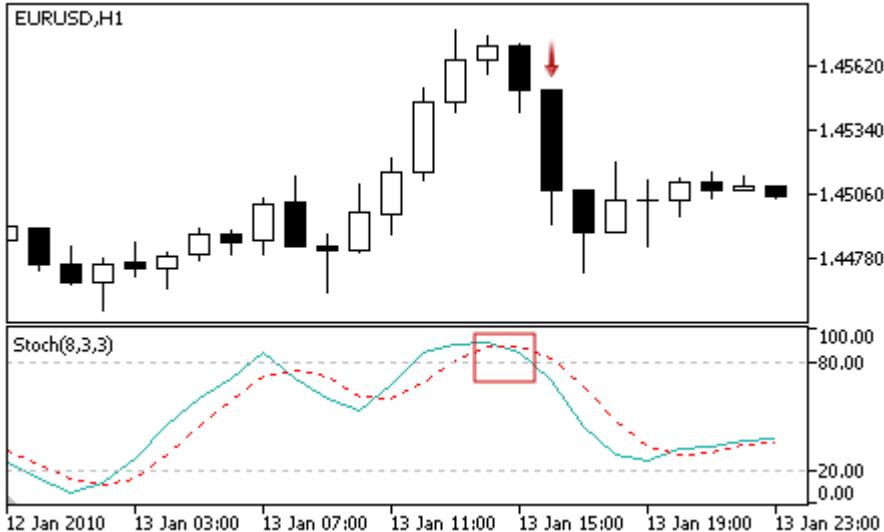
Данный модуль сигналов основан на рыночных моделях осциллятора [Stochastic](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

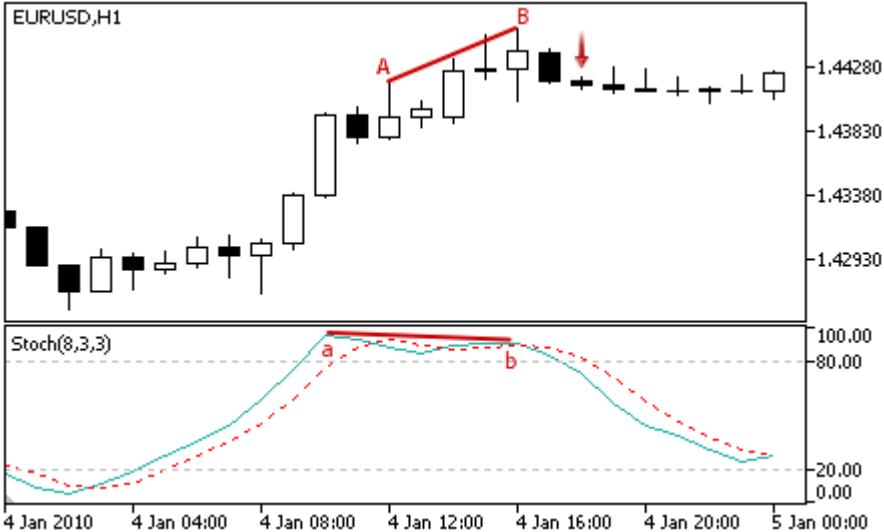
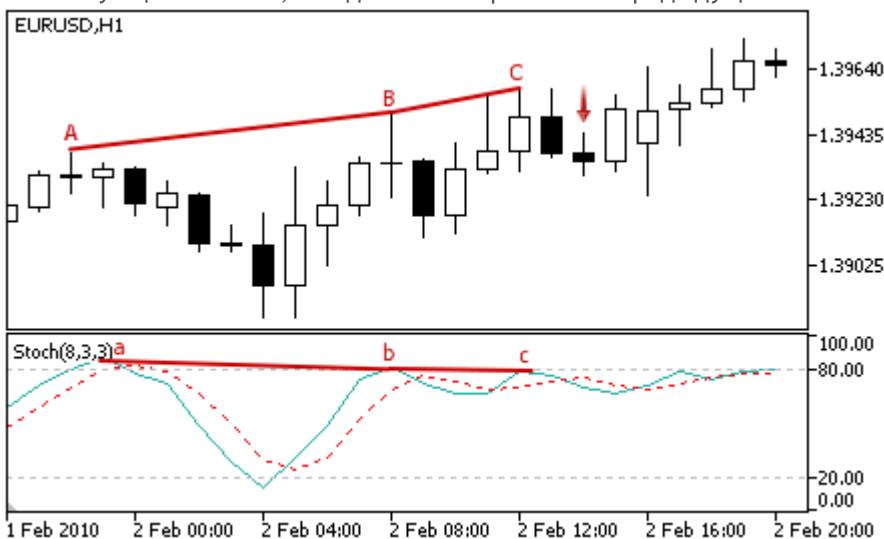
### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li><b>Разворот</b> — осциллятор развернулся вверх (осциллятор растет на анализируемом баре, а на предыдущем он падал).</li> </ul>  <p>EURUSD,H1 Stoch(8,3,3) 14 Jan 2010 14 Jan 20:00 15 Jan 00:00 15 Jan 04:00 15 Jan 08:00 15 Jan 12:00 15 Jan 16:00</p> <ul style="list-style-type: none"> <li><b>Пересечение основной и сигнальной линии</b> — на анализируемом баре основная линия выше сигнальной, а на предыдущем — ниже.</li> </ul>  <p>EURUSD,H1 Stoch(8,3,3) 31 Dec 2009 31 Dec 15:00 4 Jan 00:00 4 Jan 04:00 4 Jan 08:00 4 Jan 12:00 4 Jan 16:00</p> <ul style="list-style-type: none"> <li><b>Дивергенция</b> — первая анализируемая впадина осциллятора мельче предыдущей, а соответствующая ей впадина цены глубже предыдущей.</li> </ul>
За продажу	

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Двойная дивергенция – осциллятор сформировал три последовательных впадины, каждая из которых мельче предыдущей, а цена сформировала три соответствующие им впадины, каждая из которых глубже предыдущей.</li> </ul> 
За продажу	<ul style="list-style-type: none"> <li>Разворот – осциллятор развернулся вниз (осциллятор падает на анализируемом баре, а на предыдущем он рос).</li> </ul>

Тип сигнала	Описание условий
	<p><b>EURUSD,H1</b></p>  <p>Stoch(8,3,3)</p> <ul style="list-style-type: none"> <li>Пересечение основной и сигнальной линии – на анализируемом баре основная линия ниже сигнальной, а на предыдущем – выше.</li> </ul> <p><b>EURUSD,H1</b></p>  <p>Stoch(8,3,3)</p> <ul style="list-style-type: none"> <li>Дивергенция – первый анализируемый пик осциллятора ниже предыдущего, а соответствующий ему пик цены выше предыдущего.</li> </ul>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Двойная дивергенция – осциллятор сформировал три последовательных пика, каждый из которых ниже предыдущего, а цена сформировала три соответствующих им пика, каждый из которых выше предыдущего.</li> </ul> 
Не против покупки	Значение осциллятора на анализируемом баре растет.
Не против продажи	Значение осциллятора на анализируемом баре падает.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

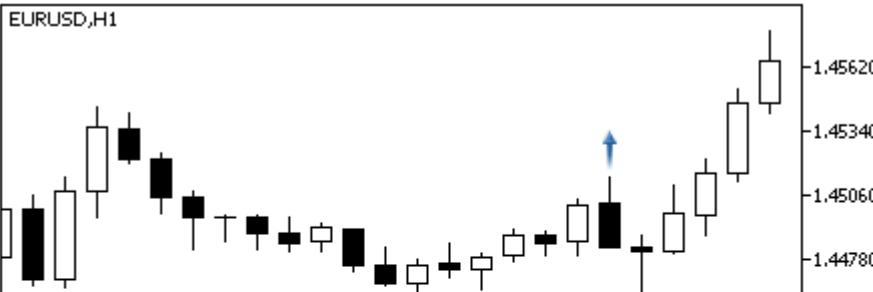
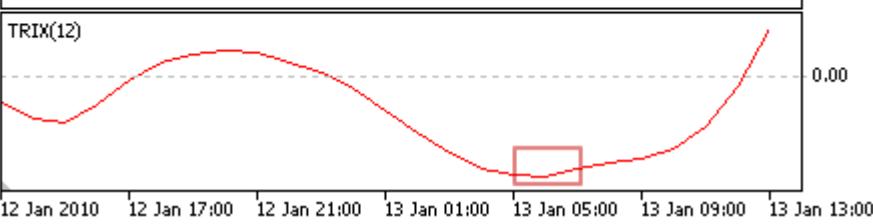
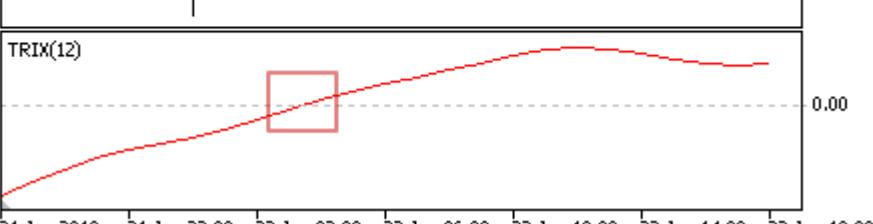
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodK	Период расчета главной линии осциллятора.
PeriodD	Период усреднения главной линии осциллятора.
PeriodSlow	Период замедления.
Applied	<a href="#">Ценовой ряд</a> , на значениях которого рассчитывается осциллятор.

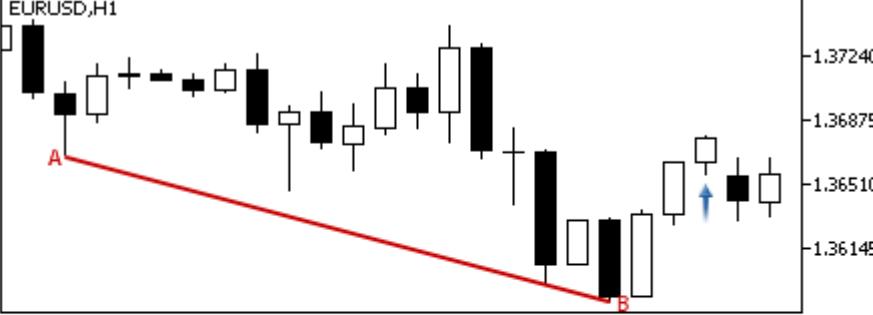
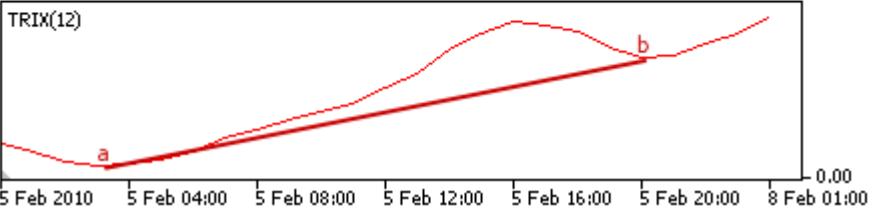
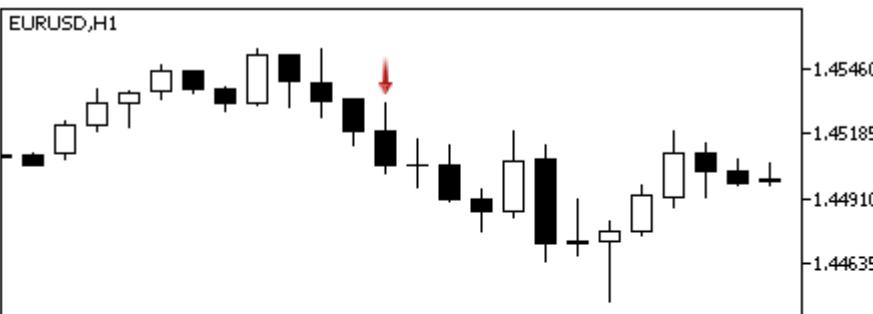
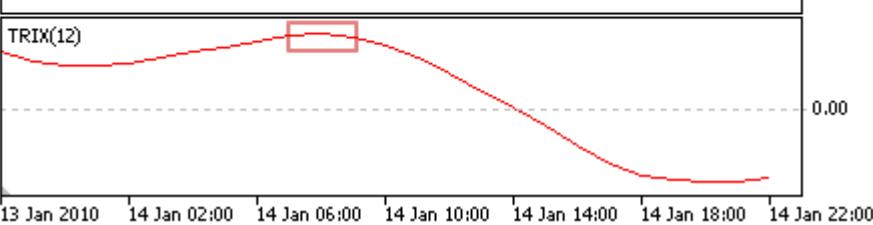
## Сигналы осциллятора Triple Exponential Average

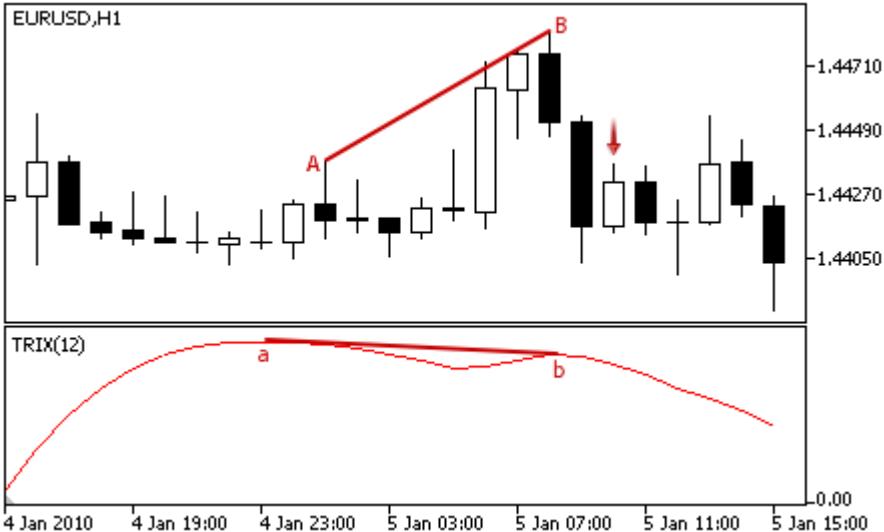
Данный модуль сигналов основан на рыночных моделях осциллятора [Triple Exponential Average](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li><b>Разворот</b> — осциллятор развернулся вверх (осциллятор растет на анализируемом баре, а на предыдущем он падал).</li> </ul>   <p>12 Jan 2010 12 Jan 17:00 12 Jan 21:00 13 Jan 01:00 13 Jan 05:00 13 Jan 09:00 13 Jan 13:00</p> <ul style="list-style-type: none"> <li><b>Пересечении нулевого уровня</b> — на анализируемом баре основная линия выше нулевого уровня, а на предыдущем — ниже.</li> </ul>   <p>21 Jan 2010 21 Jan 22:00 22 Jan 02:00 22 Jan 06:00 22 Jan 10:00 22 Jan 14:00 22 Jan 18:00</p> <ul style="list-style-type: none"> <li><b>Дивергенция</b> — первая анализируемая впадина осциллятора мельче предыдущей, а соответствующая ей впадина цены глубже предыдущей.</li> </ul>

Тип сигнала	Описание условий
	 
За продажу	<ul style="list-style-type: none"> <li>Разворот – осциллятор развернулся вниз (осциллятор падает на анализируемом баре, а на предыдущем он рос).</li> </ul>   <ul style="list-style-type: none"> <li>Пересечении нулевого уровня – на анализируемом баре основная линия ниже нулевого уровня, а на предыдущем – выше.</li> </ul>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Дивергенция — первый анализируемый пик осциллятора ниже предыдущего, а соответствующий ему пик цены выше предыдущего.</li> </ul> 
Не против покупки	Значение осциллятора на анализируемом баре растет.
Не против продажи	Значение осциллятора на анализируемом баре падает.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар — это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

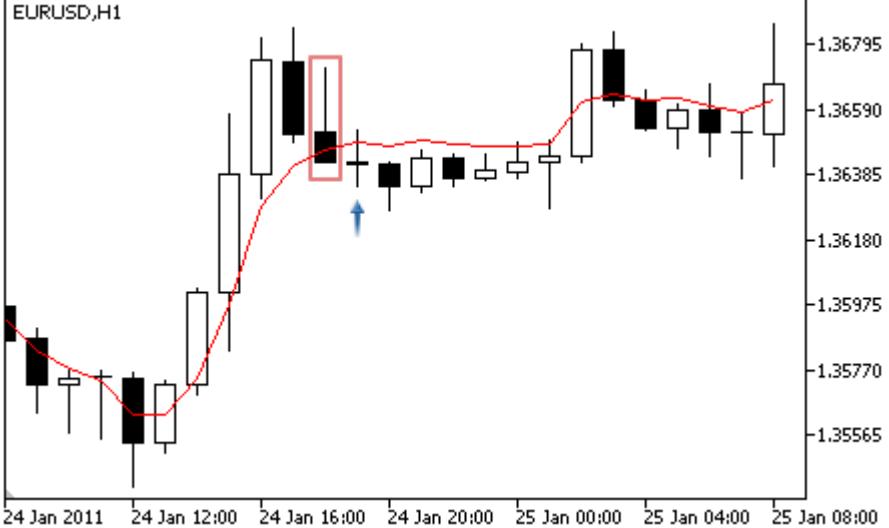
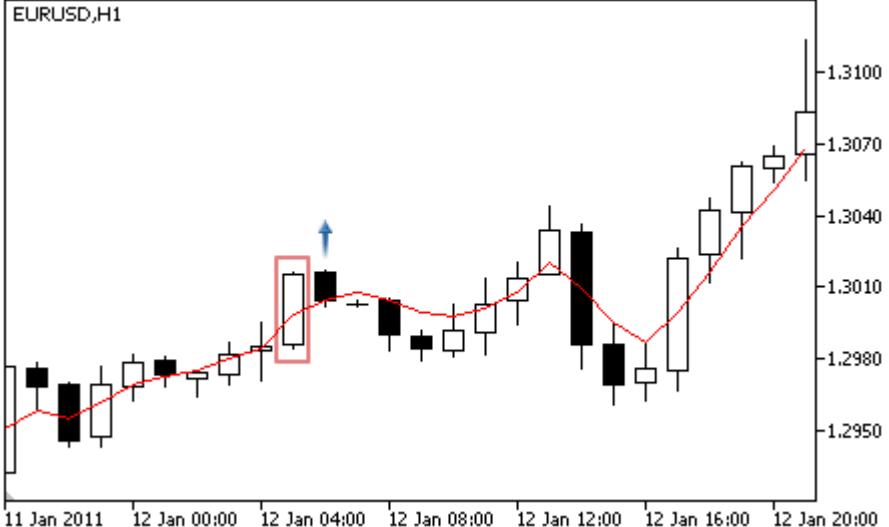
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodTriX	Период расчета осциллятора.
Applied	<u>Ценовой ряд</u> , на значениях которого рассчитывается осциллятор.

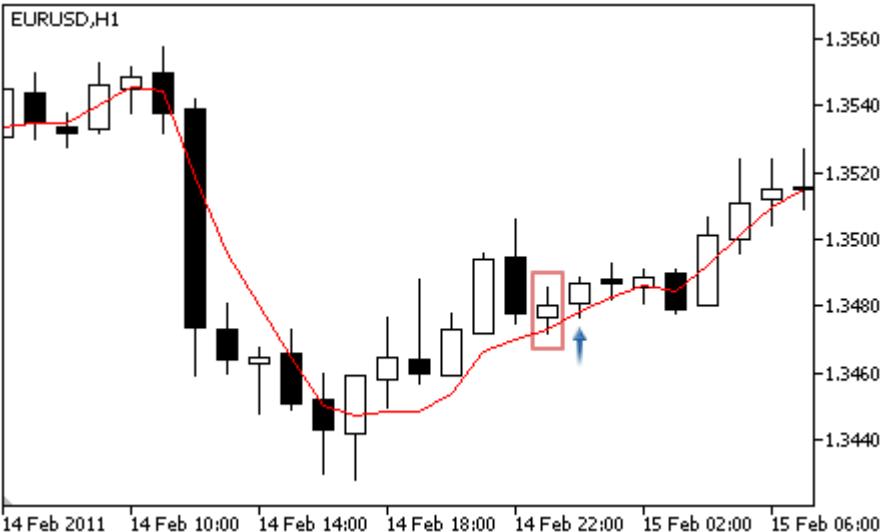
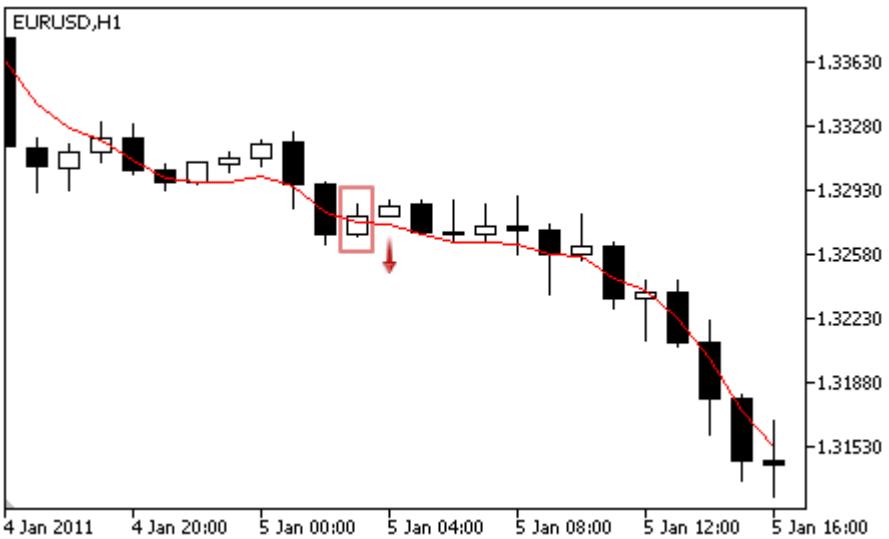
## Сигналы индикатора

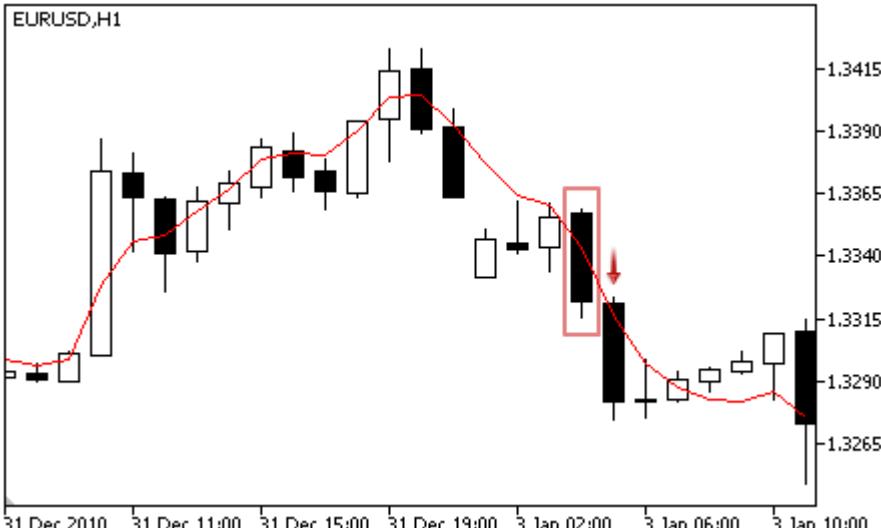
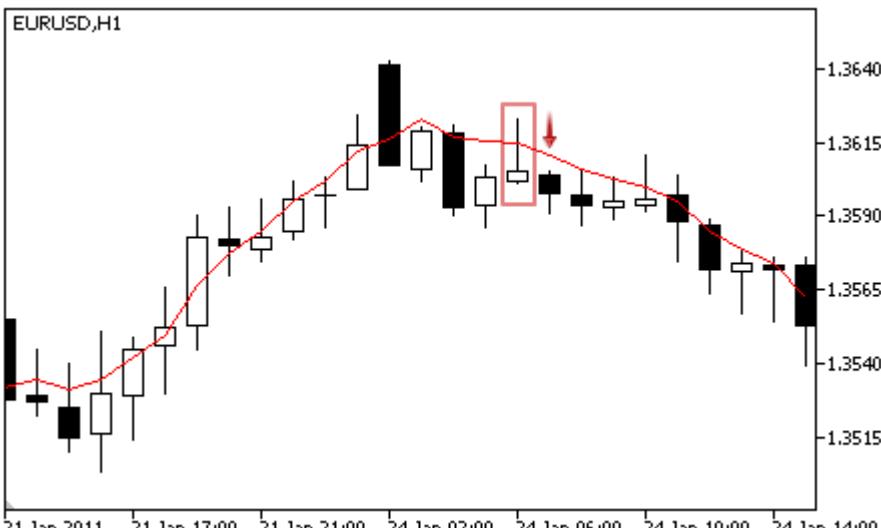
Данный модуль сигналов основан на рыночных моделях индикатора [Triple Exponential Moving Average](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li><b>Несформировавшийся прокол.</b> Цена пересекла индикатор сверху вниз(цена Open анализируемого бара выше линии индикатора, а цена Close - ниже), но индикатор растет (слабый сигнал на отбой от линии индикатора).</li> </ul>  <ul style="list-style-type: none"> <li><b>Пересечение скользящей средней.</b> Цена пересекла индикатор снизу вверх (цена Open анализируемого бара ниже линии индикатора, а цена Close - выше) и индикатор растет (сильный сигнал).</li> </ul> 

Тип сигнала	Описание условий
	<ul style="list-style-type: none"> <li>Сформировавшийся прокол. Цена пересекла индикатор нижней тенью (цены Open и Close анализируемого бара выше линии индикатора, а цена Low ниже) и индикатор растет (сигнал на отбой от линии индикатора).</li> </ul>  <p>14 Feb 2011 14 Feb 10:00 14 Feb 14:00 14 Feb 18:00 14 Feb 22:00 15 Feb 02:00 15 Feb 06:00</p>
За продажу	<ul style="list-style-type: none"> <li>Несформировавшийся прокол. Цена пересекла индикатор снизу вверх (цена Open анализируемого бара ниже линии индикатора, а цена Close - выше), но индикатор падает (слабый сигнал на отбой от линии индикатора).</li> </ul>  <p>4 Jan 2011 4 Jan 20:00 5 Jan 00:00 5 Jan 04:00 5 Jan 08:00 5 Jan 12:00 5 Jan 16:00</p> <ul style="list-style-type: none"> <li>Пересечение скользящей средней. Цена пересекла индикатор сверху вниз (цена Open анализируемого бара выше линии индикатора, а цена Close - ниже) и индикатор падает (сильный сигнал).</li> </ul>

Тип сигнала	Описание условий
	 <ul style="list-style-type: none"> <li>Сформировавшийся прокол. Цена пересекла индикатор верхней тенью (цены Open и Close анализируемого бара ниже линии индикатора, а цена High выше) и индикатор падает (сигнал на отбой от линии индикатора).</li> </ul> 
Не против покупки	Цена находится выше индикатора.
Не против продажи	Цена находится ниже индикатора.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

#### Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

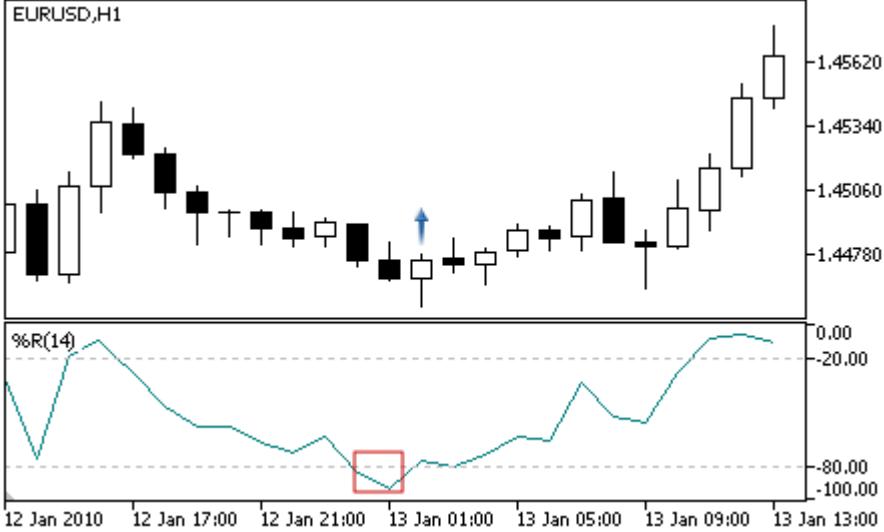
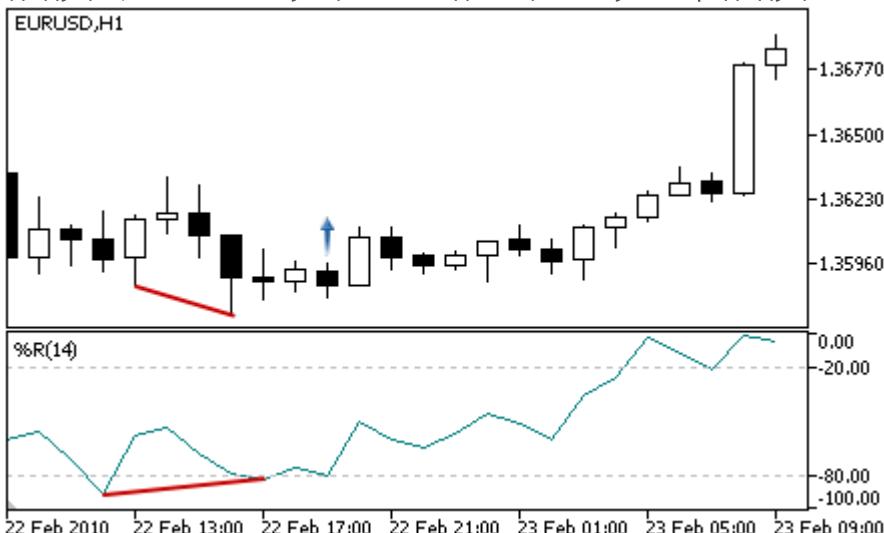
Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodMA	Период усреднения индикатора.
Shift	Смещение индикатора по оси времени (в барах).
Method	<a href="#">Метод усреднения</a> .
Applied	<a href="#">Ценовой ряд</a> , на значениях которого рассчитывается индикатор.

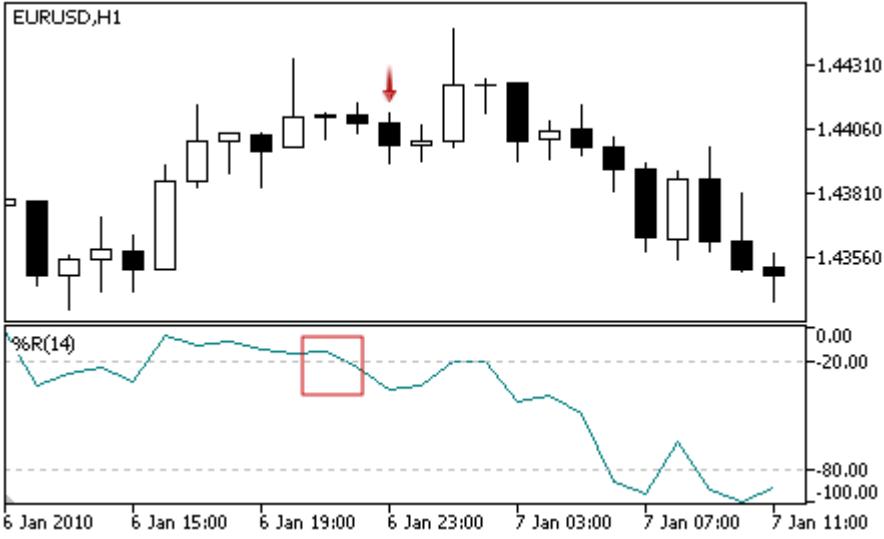
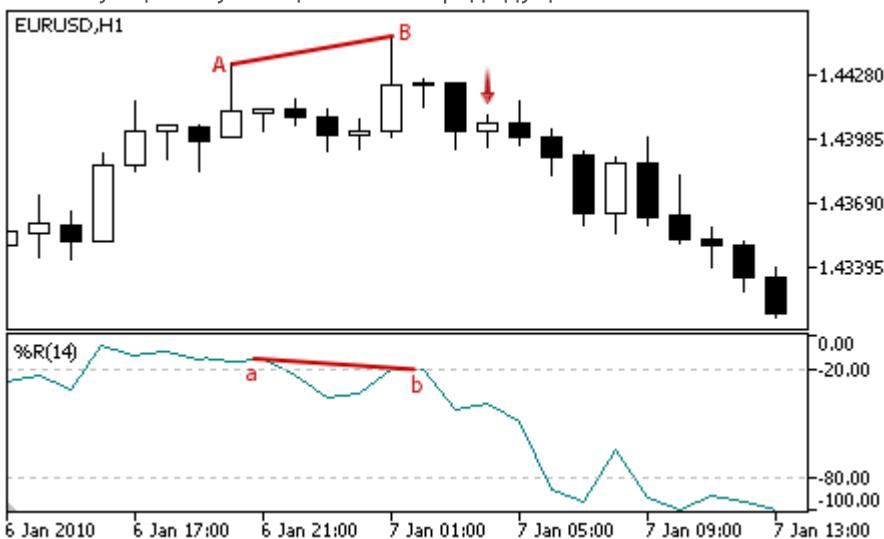
## Сигналы осциллятора Williams Percent Range

Данный модуль сигналов основан на рыночных моделях осциллятора [Williams Percent Range](#). Механизм принятия торговых решений на основе сигналов модулей описан в [отдельном разделе](#).

### Условия генерации сигналов

Ниже приведено описание условий, при которых модуль передает советнику тот или иной сигнал.

Тип сигнала	Описание условий
За покупку	<ul style="list-style-type: none"> <li>Разворот за уровнем перепроданности — осциллятор развернулся вверх и его значение на анализируемом баре находится за уровнем перепроданности (по умолчанию -80).</li> </ul> 
	<ul style="list-style-type: none"> <li>Дивергенция — первая анализируемая впадина осциллятора мельче предыдущей, а соответствующая ей впадина цены глубже предыдущей.</li> </ul> 
За продажу	<ul style="list-style-type: none"> <li>Разворот за уровнем перекупленности — осциллятор развернулся вниз и его значение на анализируемом баре находится за уровнем перекупленности (по умолчанию 20).</li> </ul>

Тип сигнала	Описание условий
	<p>умолчанию -20).</p>  <ul style="list-style-type: none"> <li>Дивергенция – первый анализируемый пик осциллятора ниже предыдущего, а соответствующий ему пик цены выше предыдущего.</li> </ul> 
Не против покупки	Значение осциллятора на анализируемом баре растет.
Не против продажи	Значение осциллятора на анализируемом баре падает.

#### Примечание

В зависимости от режима работы эксперта ("Каждый тик" или "По ценам открытия") анализируемый бар – это либо текущий бар (с индексом 0), либо последний сформировавшийся бар (с индексом 1).

Следует учитывать, что осциллятор Williams Percent Range имеет перевернутую шкалу. Его максимальное значение -100, а минимальное 0.

## Настраиваемые параметры

Данный модуль обладает следующими настраиваемыми параметрами:

Параметр	Описание
Weight	Вес сигнала модуля в интервале от 0 до 1.
PeriodWPR	Период расчета осциллятора.

## Набор готовых реализаций алгоритмов сопровождения открытых позиций

Этот раздел содержит технические детали работы с классами готовых реализаций алгоритмов сопровождения открытых позиций и описание соответствующих компонентов стандартной библиотеки MQL5.

Использование классов готовых реализаций алгоритмов сопровождения открытых позиций позволит сэкономить время при разработке (а особенно, проверке) торговых стратегий.

Стандартная библиотека MQL5 (в части набора готовых реализаций алгоритмов сопровождения открытых позиций) размещается в рабочем каталоге терминала в папке `Include\Expert\Trailing`.

Класс	Описание
<a href="#">CTrailingFixedPips</a>	Класс реализации алгоритма сопровождения открытых позиций на фиксированном "расстоянии"
<a href="#">CTrailingMA</a>	Класс реализации алгоритма сопровождения открытых позиций по значениям скользящей средней
<a href="#">CTrailingNone</a>	Класс-заглушка алгоритмов сопровождения открытых позиций
<a href="#">CTrailingPSAR</a>	Класс реализации алгоритма сопровождения открытых позиций по значениям индикатора Parabolic SAR

## Класс CTrailingFixedPips

Класс CTrailingFixedPips является классом реализации алгоритма сопровождения открытых позиций на фиксированном "расстоянии" (в пунктах).

Класс CTrailingFixedPips реализует следующий алгоритм сопровождения открытых позиций: Если уровень Stop Loss равен нулю, считается, что условие модификации ордера Stop Loss не выполнено и изменить Stop Loss позиции не предлагается, иначе проверяется факт изменения цены в сторону увеличения прибыли.

Если ордер Stop Loss у позиции уже установлен, проверяется отступ цены от Stop Loss. Если ордер StopLoss у позиции не установлен, проверяется отступ цены от цены открытия позиции. Если отступ цены больше уровня Stop Loss, то предлагается установить новую цену Stop Loss позиции.

Если условие модификации Stop Loss выполнено и уровень Take Profit не равен нулю, то предлагается установить новую цену Take Profit позиции.

Если [инициализация класса эксперта](#) произведена с флагом every\_tick=false, то эксперт будет производить торговые операции (и сопровождение торговых позиций) только при появлении нового бара на рабочем символе и таймфрейме. В этом случае установка уровня ордера Take Profit дает возможность закрыть позицию при движении цены в направлении позиции до появления нового бара.

### Описание

Класс CTrailingFixedPips реализует алгоритм сопровождения открытых позиций на указанном "расстоянии" от текущей цены (в пунктах).

### Декларация

```
class CTrailingFixedPips: public CExpertTrailing
```

### Заголовок

```
#include <Expert\Trailing\CTrailingFixedPips.mqh>
```

### Иерархия наследования

```
CObject  
CExpertBase  
CExpertTrailing  
CTrailingFixedPips
```

### Методы класса по группам

Инициализация	
<a href="#">StopLevel</a>	Устанавливает уровень Stop Loss
<a href="#">ProfitLevel</a>	Устанавливает уровень Take Profit
<a href="#">virtual ValidationSettings</a>	Проверяет корректность настроек

Методы проверки условий трейлинга	
virtual <a href="#">CheckTrailingStopLong</a>	Определяет необходимость модификации длинной позиции
virtual <a href="#">CheckTrailingStopShort</a>	Определяет необходимость модификации короткой позиции

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

## StopLevel

Устанавливает уровень сопровождения ордера Stop Loss.

```
void StopLevel(
    int stop_level           // уровень
)
```

### Параметры

*stop\_level*

[in] Значение уровня сопровождения ордера Stop Loss в "нормальных" (2/4-знаковых) пунктах.

### Примечание

Если задать уровень равным 0, то сопровождение (модификация) ордера не производится.

## ProfitLevel

Устанавливает уровень сопровождения ордера Take Profit.

```
void ProfitLevel(
    int profit_level        // уровень
)
```

### Параметры

*profit\_level*

[in] Значение уровня сопровождения ордера Take Profit в "нормальных" (2/4-знаковых) пунктах.

### Примечание

Если задать уровень равным 0, то сопровождение (модификация) ордера не производится.

## ValidationSettings

Проверяет корректность настроек.

```
virtual bool ValidationSettings()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Проверяет значения уровней сопровождения. Корректными значениями являются 0 и значения, большие уровня "Минимальный отступ в пунктах от текущей цены для установки Stop ордеров" рабочего символа.

## CheckTrailingStopLong

Определяет необходимость модификации длинной позиции.

```
virtual bool CheckTrailingStopLong(
    CPositionInfo* position,           // указатель
    double&          sl,              // ссылка для Stop Loss
    double&          tp               // ссылка для Take Profit
)
```

### Параметры

*position*

[in] Указатель на объект [CPositionInfo](#).

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

### Примечание

Если уровень сопровождения ордера Stop Loss равен нулю, условие не выполнено (уходим). Если ордер Stop Loss позиции уже установлен, его цена принимается за базовую, иначе за базовую цену принимается цена открытия позиции. Если текущая цена Bid выше базовой более чем на уровень сопровождения, то ордер Stop Loss позиции предлагается переместить на расстояние уровня сопровождения ниже текущей цены Bid. Если, при этом, уровень сопровождения ордера Take Profit не равен нулю, то предлагается переместить ордер Take Profit позиции на соответствующее расстояние выше текущей цены Bid.

## CheckTrailingStopShort

Определяет необходимость модификации короткой позиции.

```
virtual bool CheckTrailingStopShort(
    CPositionInfo* position,           // указатель
    double&          sl,              // ссылка для Stop Loss
    double&          tp               // ссылка для Take Profit
)
```

### Параметры

*position*

[in] Указатель на объект CPositionInfo.

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

### Примечание

Если уровень сопровождения ордера Stop Loss равен нулю, условие не выполнено (уходим). Если ордер Stop Loss позиции уже установлен, его цена принимается за базовую, иначе за базовую цену принимается цена открытия позиции. Если текущая цена Ask ниже базовой более чем на уровень сопровождения, то ордер Stop Loss позиции предлагается переместить на расстояние уровня сопровождения выше текущей цены Ask. Если, при этом, уровень сопровождения ордера Take Profit не равен нулю, то предлагается переместить ордер Take Profit позиции на соответствующее расстояние ниже текущей цены Ask.

## Класс CTrailingMA

Класс CTrailingMA является классом реализации алгоритма сопровождения открытых позиций по значениям скользящей средней.

### Описание

Класс CTrailingMA реализует алгоритм сопровождения открытых позиций по значениям скользящей средней с указанными параметрами на предыдущем баре.

### Декларация

```
class CTrailingMA: public CExpertTrailing
```

### Заголовок

```
#include <Expert\Trailing\TrailingMA.mqh>
```

### Иерархия наследования

[CObject](#)  
[CExpertBase](#)  
[CExpertTrailing](#)  
CTrailingMA

### Методы класса по группам

Инициализация	
<a href="#">Period</a>	Устанавливает параметр period скользящей средней
<a href="#">Shift</a>	Устанавливает параметр shift скользящей средней
<a href="#">Method</a>	Устанавливает параметр method скользящей средней
<a href="#">Applied</a>	Устанавливает параметр applied скользящей средней
<a href="#">virtual InitIndicators</a>	Инициализирует индикаторы и таймсерии
<a href="#">virtual ValidationSettings</a>	Проверяет корректность настроек
<b>Методы проверки условий трейлинга</b>	
<a href="#">virtual CheckTrailingStopLong</a>	Определяет необходимость модификации длинной позиции
<a href="#">virtual CheckTrailingStopShort</a>	Определяет необходимость модификации короткой позиции

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#),  
[RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#)

## Period

Устанавливает параметр period для скользящей средней.

```
void Period(  
    int period // параметр  
)
```

### Параметры

*period*

[in] Значение параметра period для скользящей средней.

## Shift

Устанавливает параметр shift для скользящей средней.

```
void Shift(
    int shift           // параметр
)
```

### Параметры

*shift*

[in] Значение параметра shift для скользящей средней.

## Method

Устанавливает параметр method для скользящей средней.

```
void Method(
    ENUM_MA_METHOD method      // параметр
)
```

### Параметры

*method*

[in] Значение параметра method (из перечисления [ENUM\\_MA\\_METHOD](#)) для скользящей средней.

## Applied

Устанавливает параметр applied для скользящей средней.

```
void Applied(
    ENUM_APPLIED_PRICE applied           // параметр
)
```

### Параметры

*applied*

[in] Значение параметра applied (из перечисления [ENUM\\_APPLIED\\_PRICE](#)) для скользящей средней.

## InitIndicators

Инициализирует индикаторы и таймсерии.

```
virtual bool InitIndicators(
    CIndicators* indicators          // указатель
)
```

### Параметры

*indicators*

[in] Указатель на коллекцию индикаторов и таймсерий, член класса [CExpert](#).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ValidationSettings

Проверяет корректность настроек.

```
virtual bool ValidationSettings()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Проверяет значение периода скользящей средней. Корректными являются значения больше 0.

## CheckTrailingStopLong

Определяет необходимость модификации длинной позиции.

```
virtual bool CheckTrailingStopLong(
    CPositionInfo* position,           // указатель
    double&          sl,              // ссылка
    double&          tp               // ссылка
)
```

### Параметры

*position*

[in] Указатель на объект [CPositionInfo](#).

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

### Примечание

Вычисляется значение максимально близкого уровня установки Stop Loss. Вычисляется новый уровень установки Stop Loss (исходя из значения средней скользящей на предыдущем баре). Если ордер Stop Loss позиции уже установлен, его цена принимается за базовую, иначе за базовую цену принимается цена открытия позиции. Если новый уровень установки Stop Loss выше базовой цены и ниже максимально близкого уровня установки Stop Loss, то ордер Stop Loss позиции предлагается переместить на новый уровень.

## CheckTrailingStopShort

Определяет необходимость модификации короткой позиции.

```
virtual bool CheckTrailingStopShort(
    CPositionInfo* position,           // указатель
    double&          sl,              // ссылка
    double&          tp               // ссылка
)
```

### Параметры

*position*

[in] Указатель на объект [CPositionInfo](#).

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

### Примечание

Вычисляется значение максимально близкого уровня установки Stop Loss. Вычисляется новый уровень установки Stop Loss (исходя из значения средней скользящей на предыдущем баре с учетом спреда). Если ордер Stop Loss позиции уже установлен, его цена принимается за базовую, иначе за базовую цену принимается цена открытия позиции. Если новый уровень установки Stop Loss ниже базовой цены и выше максимально близкого уровня установки Stop Loss, то ордер Stop Loss позиции предлагается переместить на новый уровень.

## Класс CTrailingNone

Класс CTrailingNone является классом-заглушкой. Его необходимо использовать для инициализации объекта трейлинга в классе CExpert, в том случае, если торговая стратегия не использует сопровождение открытых позиций.

### Описание

Класс CTrailingNone не реализует никаких алгоритмов сопровождения открытых позиций. Методы проверки условий трейлинга всегда возвращают false.

### Декларация

```
class CTrailingNone: public CExpertTrailing
```

### Заголовок

```
#include <Expert\Trailing\TrailingNone.mqh>
```

### Иерархия наследования

```
CObject
  CExpertBase
    CExpertTrailing
      CTrailingNone
```

### Методы класса по группам

Методы проверки условий трейлинга	
virtual <a href="#">CheckTrailingStopLong</a>	Метод-заглушка проверки необходимости модификации длинной позиции
virtual <a href="#">CheckTrailingStopShort</a>	Метод-заглушка проверки необходимости модификации короткой позиции

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [ValidationSettings](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

#### Методы унаследованные от CExpertTrailing

[CheckTrailingStopLong](#), [CheckTrailingStopShort](#)

## CheckTrailingStopLong

Определяет необходимость модификации длинной позиции.

```
virtual bool CheckTrailingStopLong(
    CPositionInfo* position,           // указатель
    double&          sl,              // ссылка
    double&          tp               // ссылка
)
```

### Параметры

*position*

[in] Указатель на объект [CPositionInfo](#).

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

### Примечание

Метод всегда возвращает false.

## CheckTrailingStopShort

Определяет необходимость модификации короткой позиции.

```
virtual bool CheckTrailingStopShort(
    CPositionInfo* position,           // указатель
    double&          sl,              // ссылка
    double&          tp               // ссылка
)
```

### Параметры

*position*

[in] Указатель на объект [CPositionInfo](#).

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

### Примечание

Метод всегда возвращает false.

## Класс CTrailingPSAR

Класс CTrailingPSAR является классом реализации алгоритма сопровождения открытых позиций по значениям индикатора Parabolic SAR.

### Описание

Класс CTrailingPSAR реализует алгоритм сопровождения открытых позиций по значениям (на предыдущем баре) индикатора Parabolic SAR с указанными параметрами.

### Декларация

```
class CTrailingPSAR: public CExpertTrailing
```

### Заголовок

```
#include <Expert\Trailing\TrailingParabolicSAR.mqh>
```

### Иерархия наследования

```
CObject  
CExpertBase  
CExpertTrailing  
CTrailingPSAR
```

### Методы класса по группам

Инициализация	
<a href="#">Step</a>	Устанавливает параметр step индикатора Parabolic SAR
<a href="#">Maximum</a>	Устанавливает параметр maximum индикатора Parabolic SAR
<a href="#">virtual InitIndicators</a>	Инициализирует индикаторы и таймсерии
<b>Методы проверки условий трейлинга</b>	
<a href="#">virtual CheckTrailingStopLong</a>	Определяет необходимость модификации длинной позиции
<a href="#">virtual CheckTrailingStopShort</a>	Определяет необходимость модификации короткой позиции

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Методы унаследованные от CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [ValidationSettings](#), [SetPriceSeries](#), [SetOtherSeries](#)



## Step

Устанавливает параметр step индикатора Parabolic SAR.

```
void Step(  
    double step // параметр  
)
```

### Параметры

*step*

[in] Значение параметра step для индикатора Parabolic SAR.

## Maximum

Устанавливает параметр maximum индикатора Parabolic SAR.

```
void Maximum(  
    double maximum           // параметр  
)
```

### Параметры

*maximum*

[in] Значение параметра maximum для индикатора Parabolic SAR.

## InitIndicators

Инициализирует индикаторы и таймсерии.

```
virtual bool InitIndicators(
    CIndicators* indicators          // указатель
)
```

### Параметры

*indicators*

[in] Указатель на коллекцию индикаторов и таймсерий, член класса [CExpert](#).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CheckTrailingStopLong

Определяет необходимость модификации длинной позиции.

```
virtual bool CheckTrailingStopLong(
    CPositionInfo* position,           // указатель
    double&          sl,              // ссылка
    double&          tp               // ссылка
)
```

### Параметры

*position*

[in] Указатель на объект [CPositionInfo](#).

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

### Примечание

Вычисляется значение максимально близкого уровня установки Stop Loss. Вычисляется новый уровень установки Stop Loss (исходя из значения индикатора Parabolic SAR на предыдущем баре). Если ордер Stop Loss позиции уже установлен, его цена принимается за базовую, иначе за базовую цену принимается цена открытия позиции. Если новый уровень установки Stop Loss выше базовой цены и ниже максимально близкого уровня установки Stop Loss, то ордер Stop Loss позиции предлагается переместить на новый уровень.

## CheckTrailingStopShort

Определяет необходимость модификации короткой позиции.

```
virtual bool CheckTrailingStopShort(
    CPositionInfo* position,           // указатель
    double&          sl,              // ссылка
    double&          tp               // ссылка
)
```

### Параметры

*position*

[in] Указатель на объект [CPositionInfo](#).

*sl*

[in][out] Ссылка на переменную для размещения цены Stop Loss.

*tp*

[in][out] Ссылка на переменную для размещения цены Take Profit.

### Возвращаемое значение

true - в случае выполнения условия, иначе - false.

### Примечание

Вычисляется значение максимально близкого уровня установки Stop Loss. Вычисляется новый уровень установки Stop Loss (исходя из значения индикатора Parabolic SAR на предыдущем баре с учетом спреда). Если ордер Stop Loss позиции уже установлен, его цена принимается за базовую, иначе за базовую цену принимается цена открытия позиции. Если новый уровень установки Stop Loss ниже базовой цены и выше максимально близкого уровня установки Stop Loss, то ордер Stop Loss позиции предлагается переместить на новый уровень.

## Набор готовых реализаций алгоритмов управления капиталом и рисками

Этот раздел содержит технические детали работы с классами готовых реализаций алгоритмов управления капиталом и рисками и описание соответствующих компонентов стандартной библиотеки MQL5.

Использование классов готовых реализаций алгоритмов управления капиталом и рисками позволит сэкономить время при разработке (а особенно, проверке) торговых стратегий.

Стандартная библиотека MQL5 (в части набора готовых реализаций алгоритмов управления капиталом и рисками) размещается в рабочем каталоге терминала в папке `Include\Expert\Money`.

Класс	Описание
<a href="#">CMoneyFixedLot</a>	Класс реализации алгоритма для входа в рынок фиксированным лотом, заданным при настройке
<a href="#">CMoneyFixedMargin</a>	Класс реализации алгоритма для входа в рынок фиксированным уровнем маржи, заданным при настройке
<a href="#">CMoneyFixedRisk</a>	Класс реализации алгоритма для входа в рынок фиксированным уровнем риска, заданным при настройке
<a href="#">CMoneyNone</a>	Класс реализации алгоритма для входа в рынок минимальным лотом
<a href="#">CMoneySizeOptimized</a>	Класс реализации алгоритма для входа в рынок с объемом, определяемом результатами предыдущих сделок

## Класс CMoneyFixedLot

Класс CMoneyFixedLot является классом для реализации алгоритма торговли с фиксированным лотом.

### Описание

Класс CMoneyFixedLot реализует алгоритм для входа в рынок фиксированным лотом, заданным при настройке.

### Декларация

```
class CMoneyFixedLot: public CExpertMoney
```

### Заголовок

```
#include <Expert\Money\MoneyFixedLot.mqh>
```

### Иерархия наследования

```
CObject
  CExpertBase
    CExpertMoney
      CMoneyFixedLot
```

### Методы класса по группам

Инициализация	
<a href="#">Lots</a>	Устанавливает торговый объем
<a href="#">virtual ValidationSettings</a>	Проверяет корректность настроек
Методы управления капиталом и рисками	
<a href="#">virtual CheckOpenLong</a>	Определяет объем для открытия длинной позиции
<a href="#">virtual CheckOpenShort</a>	Определяет объем для открытия короткой позиции

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Методы унаследованные от CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

### Методы унаследованные от CExpertMoney

[Percent](#), [CheckReverse](#), [CheckClose](#)



## Lots

Устанавливает торговый объем (в лотах).

```
void Lots(
    double lots           // количество лотов
)
```

### Параметры

*lots*

[in] Торговый объем.

## ValidationSettings

Проверяет корректность настроек.

```
virtual bool ValidationSettings()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Проверяет фиксированный лот, задаваемый при настройке, на предмет попадания в интервал допустимых значений и кратность шагу изменения.

## CheckOpenLong

Определяет объем для открытия длинной позиции.

```
virtual double CheckOpenLong(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Предполагаемая цена открытия.

*sl*

[in] Предполагаемая цена ордера Stop Loss.

### Возвращаемое значение

Объем для открытия длинной позиции.

### Примечание

Метод всегда предлагает фиксированный лот, заданный при настройке.

## CheckOpenShort

Определяет объем для открытия короткой позиции.

```
virtual double CheckOpenShort(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Предполагаемая цена открытия.

*sl*

[in] Предполагаемая цена ордера Stop Loss.

### Возвращаемое значение

Объем для открытия короткой позиции.

### Примечание

Метод всегда предлагает фиксированный лот, заданный при настройке.

## Класс CMoneyFixedMargin

Класс CMoneyFixedMargin является классом для реализации алгоритма торговли с фиксированным уровнем маржи.

### Описание

Класс CMoneyFixedMargin реализует алгоритм для входа в рынок фиксированным уровнем маржи, заданным при настройке.

### Декларация

```
class CMoneyFixedMargin: public CExpertMoney
```

### Заголовок

```
#include <Expert\Money\MoneyFixedMargin.mqh>
```

### Иерархия наследования

```
CObject
CExpertBase
CExpertMoney
CMoneyFixedMargin
```

### Методы класса по группам

Методы управления капиталом и рисками	
virtual <a href="#">CheckOpenLong</a>	Определяет объем для открытия длинной позиции
virtual <a href="#">CheckOpenShort</a>	Определяет объем для открытия короткой позиции

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

#### Методы унаследованные от CExpertMoney

[Percent](#), [ValidationSettings](#), [CheckReverse](#), [CheckClose](#)

## CheckOpenLong

Определяет объем для открытия длинной позиции.

```
virtual double CheckOpenLong(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Предполагаемая цена открытия.

*sl*

[in] Предполагаемая цена ордера Stop Loss.

### Возвращаемое значение

Объём для открытия длинной позиции.

### Примечание

Метод определяет объем для открытия длинной позиции с фиксированным уровнем маржи, заданным при настройке. Уровень маржи определяется параметром Percent базового класса [CExpertMoney](#).

## CheckOpenShort

Определяет объем для открытия короткой позиции.

```
virtual double CheckOpenShort(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Предполагаемая цена открытия.

*sl*

[in] Предполагаемая цена ордера Stop Loss.

### Возвращаемое значение

Объем для открытия короткой позиции.

### Примечание

Метод определяет объем для открытия короткой позиции с фиксированным уровнем маржи, заданным при настройке. Уровень маржи определяется параметром Percent базового класса [CExpertMoney](#).

## Класс CMoneyFixedRisk

Класс CMoneyFixedRisk является классом для реализации алгоритма торговли с фиксированным уровнем риска.

### Описание

Класс CMoneyFixedRisk реализует алгоритм для входа в рынок фиксированным уровнем риска, заданным при настройке.

### Декларация

```
class CMoneyFixedRisk: public CExpertMoney
```

### Заголовок

```
#include <Expert\Money\MoneyFixedRisk.mqh>
```

### Иерархия наследования

```
CObject  
CExpertBase  
CExpertMoney  
CMoneyFixedRisk
```

### Методы класса по группам

Методы управления капиталом и рисками	
virtual <a href="#">CheckOpenLong</a>	Определяет объем для открытия длинной позиции
virtual <a href="#">CheckOpenShort</a>	Определяет объем для открытия короткой позиции

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

#### Методы унаследованные от CExpertMoney

[Percent](#), [ValidationSettings](#), [CheckReverse](#)

## CheckOpenLong

Определяет объем для открытия длинной позиции.

```
virtual double CheckOpenLong(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Предполагаемая цена открытия.

*sl*

[in] Предполагаемая цена ордера Stop Loss.

### Возвращаемое значение

Объем для открытия длинной позиции.

### Примечание

Метод определяет объем для открытия длинной позиции с фиксированным уровнем риска, заданным при настройке. Уровень маржи определяется параметром Percent базового класса [CExpertMoney](#).

## CheckOpenShort

Определяет объем для открытия короткой позиции.

```
virtual double CheckOpenShort(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Предполагаемая цена открытия.

*sl*

[in] Предполагаемая цена ордера Stop Loss.

### Возвращаемое значение

Объем для открытия короткой позиции.

### Примечание

Метод определяет объем для открытия короткой позиции с фиксированным уровнем риска, заданным при настройке. Уровень маржи определяется параметром Percent базового класса [CExpertMoney](#).

## Класс CMoneyNone

Класс CMoneyNone является классом для реализации "отсутствия" управления капиталом и рисками.

### Описание

Класс CMoneyNone реализует алгоритм для входа в рынок минимальным лотом.

### Декларация

```
class CMoneyNone: public CExpertMoney
```

### Заголовок

```
#include <Expert\Money\MoneyNone.mqh>
```

### Иерархия наследования

```
CObject
  CExpertBase
    CExpertMoney
      CMoneyNone
```

### Методы класса по группам

Инициализация	
virtual <a href="#">ValidationSettings</a>	Проверяет корректность настроек
Методы управления капиталом и рисками	
virtual <a href="#">CheckOpenLong</a>	Определяет объем для открытия длинной позиции
virtual <a href="#">CheckOpenShort</a>	Определяет объем для открытия короткой позиции

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

#### Методы унаследованные от CExpertMoney

[Percent](#), [CheckReverse](#), [CheckClose](#)

## ValidationSettings

Проверяет корректность настроек.

```
virtual bool ValidationSettings()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Метод всегда возвращает true.

## CheckOpenLong

Определяет объём для открытия длинной позиции.

```
virtual double CheckOpenLong(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Предполагаемая цена открытия.

*sl*

[in] Предполагаемая цена ордера Stop Loss.

### Возвращаемое значение

Объём для открытия длинной позиции.

### Примечание

Метод всегда предлагает минимальный лот.

## CheckOpenShort

Определяет объем для открытия короткой позиции.

```
virtual double CheckOpenShort(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Предполагаемая цена открытия.

*sl*

[in] Предполагаемая цена ордера Stop Loss.

### Возвращаемое значение

Объем для открытия короткой позиции.

### Примечание

Метод всегда предлагает минимальный лот.

## Класс CMoneySizeOptimized

Класс CMoneySizeOptimized является классом для реализации алгоритма управления капиталом и рисками с учетом результатов предыдущих сделок.

### Описание

Класс CMoneySizeOptimized реализует алгоритм для входа в рынок с объемом, определяемым результатами предыдущих сделок.

### Декларация

```
class CMoneySizeOptimized: public CExpertMoney
```

### Заголовок

```
#include <Expert\Money\MoneySizeOptimized.mqh>
```

### Иерархия наследования

```
CObject
  CExpertBase
    CExpertMoney
      CMoneySizeOptimized
```

### Методы класса по группам

<b>Инициализация</b>	
<a href="#">DecreaseFactor</a>	Устанавливает значение параметра
<a href="#">virtual ValidationSettings</a>	Проверяет корректность настроек
<b>Методы управления капиталом и рисками</b>	
<a href="#">virtual CheckOpenLong</a>	Определяет объем для открытия длинной позиции
<a href="#">virtual CheckOpenShort</a>	Определяет объем для открытия короткой позиции

### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

### Методы унаследованные от CExpertBase

[InitPhase](#), [TrendType](#), [UsedSeries](#), [EveryTick](#), [Open](#), [High](#), [Low](#), [Close](#), [Spread](#), [Time](#), [TickVolume](#), [RealVolume](#), [Init](#), [Symbol](#), [Period](#), [Magic](#), [SetMarginMode](#), [SetPriceSeries](#), [SetOtherSeries](#), [InitIndicators](#)

### Методы унаследованные от CExpertMoney

[Percent](#), [CheckReverse](#), [CheckClose](#)



## DecreaseFactor

Устанавливает параметр "Фактор снижения".

```
void DecreaseFactor(
    double decrease_factor           // фактор снижения
)
```

### Параметры

*decrease\_factor*

[in] Значение параметра "Фактор снижения".

### Примечание

Параметр задает коэффициент уменьшения объема открываемой позиции (относительно объема предыдущей позиции), если перед этим была серия убыточных сделок.

## ValidationSettings

Проверяет корректность настроек.

```
virtual bool ValidationSettings()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Проверяет на "неотрицательность" значение параметра "Фактор снижения", задаваемое при настройке.

## CheckOpenLong

Определяет объем для открытия длинной позиции.

```
virtual double CheckOpenLong(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Предполагаемая цена открытия.

*sl*

[in] Предполагаемая цена ордера Stop Loss.

### Возвращаемое значение

Объем для открытия длинной позиции.

### Примечание

Метод определяет объем для открытия длинной позиции с учетом результатов предыдущих сделок.

## CheckOpenShort

Определяет объем для открытия короткой позиции.

```
virtual double CheckOpenShort(
    double price,           // цена
    double sl                // цена Stop Loss
)
```

### Параметры

*price*

[in] Предполагаемая цена открытия.

*sl*

[in] Предполагаемая цена ордера Stop Loss.

### Возвращаемое значение

Объем для открытия короткой позиции.

### Примечание

Метод определяет объем для открытия короткой позиции с учетом результатов предыдущих сделок.

## Классы для создания панелей индикации и диалогов управления

Этот раздел содержит технические детали работы с классами для создания панелей индикации и диалогов управления и описание соответствующих компонентов Стандартной библиотеки MQL5.

Использование классов для создания панелей индикации и диалогов управления позволит сэкономить время при разработке собственных интерактивных MQL5-приложений, каковыми могут быть эксперты и индикаторы.

Стандартная библиотека MQL5 (в части набора классов для создания панелей индикации и диалогов управления) размещается в рабочем каталоге терминала в папке MQL5\Include\Controls.

Пример эксперта, иллюстрирующего работу данных классов, находится в папке MQL5\Expert\Examples\Controls.

Вспомогательные структуры	Описание
<a href="#">CRect</a>	Структура прямоугольной области графика
<a href="#">CDateTime</a>	Структура для работы с датой и временем

Базовые классы	Описание
<a href="#">CWnd</a>	Базовый класс для всех элементов управления
<a href="#">CWndObj</a>	Базовый класс для создания панелей индикации и диалогов управления
<a href="#">CWndContainer</a>	Базовый класс для комбинированных элементов управления

Простые элементы управления	Описание
<a href="#">CLabel</a>	Элемент управления на основе объекта графика "Текстовая метка"
<a href="#">CBmpButton</a>	Элемент управления на основе объекта графика "Графическая метка"
<a href="#">CButton</a>	Элемент управления на основе объекта графика "Кнопка"
<a href="#">CEdit</a>	Элемент управления на основе объекта графика "Поле ввода"
<a href="#">CPanel</a>	Элемент управления на основе объекта графика "Прямоугольная метка"
<a href="#">CPicture</a>	Элемент управления на основе объекта графика "Графическая метка"

Комбинированные элементы управления	Описание
<a href="#"><u>CScroll</u></a>	Базовый класс полосы прокрутки
<a href="#"><u>CScrollV</u></a>	Вертикальная полоса прокрутки
<a href="#"><u>CScrollH</u></a>	Горизонтальная полоса прокрутки
<a href="#"><u>CWndClient</u></a>	Базовый класс элемента управления, имеющего полосы прокрутки
<a href="#"><u>CListView</u></a>	Просмотр списка
<a href="#"><u>CComboBox</u></a>	Поле с выпадающим списком
<a href="#"><u>CCheckBox</u></a>	Переключатель
<a href="#"><u>CCheckGroup</u></a>	Групповой переключатель с независимой фиксацией
<a href="#"><u>CRadioButton</u></a>	Элемент группового переключателя с зависимой фиксацией
<a href="#"><u>CRadioGroup</u></a>	Групповой переключатель с зависимой фиксацией
<a href="#"><u>CSpinEdit</u></a>	Поле инкремента/декремента
<a href="#"><u>CDialog</u></a>	Диалог
<a href="#"><u>CAppDialog</u></a>	Главный диалог MQL5-программы

## Класс CRect

Класс CRect является реализацией класса прямоугольной области графика.

### Описание

Класс CRect является программной реализацией прямоугольной области графика в декартовых координатах.

### Декларация

```
class CRect
```

### Заголовок

```
#include <Controls\Rect.mqh>
```

### Методы класса по группам

Свойства	
<a href="#"><u>Left</u></a>	Получает/устанавливает координату X левой верхней точки области
<a href="#"><u>Top</u></a>	Получает/устанавливает координату Y левой верхней точки области
<a href="#"><u>Right</u></a>	Получает/устанавливает координату X правой нижней точки области
<a href="#"><u>Bottom</u></a>	Получает/устанавливает координату Y правой нижней точки области
<a href="#"><u>Width</u></a>	Получает/устанавливает ширину области
<a href="#"><u>Height</u></a>	Получает/устанавливает высоту области
<a href="#"><u>SetBound</u></a>	Устанавливает новые координаты области
<a href="#"><u>Move</u></a>	Осуществляет абсолютное перемещение координат области
<a href="#"><u>Shift</u></a>	Осуществляет относительное перемещение (сдвиг) координат области
<a href="#"><u>Contains</u></a>	Проверяет факт нахождения точки внутри области
<b>Дополнительные методы</b>	
<a href="#"><u>Format</u></a>	Получает координаты области в виде строки

## Left (метод Get)

Получает координату X левой верхней точки прямоугольной области.

```
int Left()
```

### Возвращаемое значение

Координата X левой верхней точки.

## Left (метод Set)

Устанавливает координату X левой верхней точки прямоугольной области.

```
void Left(  
    const int x // координата x  
)
```

### Параметры

x

[in] Новое значение координаты X левой верхней точки.

### Возвращаемое значение

Нет.

## Top (метод Get)

Получает координату Y левой верхней точки прямоугольной области.

```
int Top()
```

### Возвращаемое значение

Координата Y левой верхней точки.

## Top (метод Set)

Устанавливает координату Y левой верхней точки прямоугольной области.

```
void Top(  
    const int y // координата y  
)
```

### Параметры

*y*

[in] Новое значение координаты Y левой верхней точки.

### Возвращаемое значение

Нет.

## Right (метод Get)

Получает координату X правой нижней точки прямоугольной области.

```
int Right()
```

### Возвращаемое значение

Координата X правой нижней точки.

## Right (метод Set)

Устанавливает координату X правой нижней точки прямоугольной области.

```
void Right(  
    const int x // координата x  
)
```

### Параметры

x

[in] Новое значение координаты X правой нижней точки.

### Возвращаемое значение

Нет.

## Bottom (метод Get)

Получает координату Y правой нижней точки прямоугольной области.

```
int Bottom()
```

### Возвращаемое значение

Координата Y правой нижней точки.

## Bottom (метод Set)

Устанавливает координату Y правой нижней точки прямоугольной области.

```
void Bottom(  
    const int y // координата y  
)
```

### Параметры

*y*

[in] Новое значение координаты Y правой нижней точки.

### Возвращаемое значение

Нет.

## Width (метод Get)

Получает ширину прямоугольной области.

```
int Width()
```

### Возвращаемое значение

Ширина прямоугольной области.

## Width (метод Set)

Устанавливает ширину прямоугольной области.

```
virtual bool Width(  
    const int w // ширина  
)
```

### Параметры

w

[in] Новое значение ширины.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Height (метод Get)

Получает высоту прямоугольной области.

```
int Height()
```

### Возвращаемое значение

Высота прямоугольной области.

## Height (метод Set)

Устанавливает высоту прямоугольной области.

```
virtual bool Height(  
    const int h // высота  
)
```

### Параметры

*h*

[in] Новое значение высоты.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## SetBound

Устанавливает новые параметры прямоугольной области из координат класса CRect.

```
void SetBound(
    const & CRect rect      // класс прямоугольной области
)
```

### Возвращаемое значение

Нет.

## SetBound

Устанавливает новые параметры прямоугольной области.

```
void SetBound(
    const int l      // left
    const int t      // top
    const int r      // right
    const int b      // bottom
)
```

### Параметры

*l*

[in] Координата X левой верхней точки.

*t*

[in] Координата Y левой верхней точки.

*r*

[in] Координата X правой нижней точки.

*b*

[in] Координата Y правой нижней точки.

### Возвращаемое значение

Нет.

## Move

Осуществляет абсолютное перемещение координат прямоугольной области.

```
void Move(
    const int x,           // координата X
    const int y            // координата Y
)
```

### Параметры

*x*

[in] Новое значение координаты X.

*y*

[in] Новое значение координаты Y.

### Возвращаемое значение

Нет.

## Shift

Осуществляет относительное перемещение (сдвиг) координат прямоугольной области.

```
void Shift(
    const int dx,      // смещение по X
    const int dy       // смещение по Y
)
```

### Параметры

*dx*

[in] Относительное смещение по координате X.

*dy*

[in] Относительное смещение по координате Y.

### Возвращаемое значение

Нет.

## Contains

Проверяет факт нахождения точки с заданными координатами внутри прямоугольной области.

```
bool Contains(  
    const int x, // координата X  
    const int y // координата Y  
)
```

### Параметры

*x*

[in] Координата X точки.

*y*

[in] Координата Y точки.

### Возвращаемое значение

true, если точка находится внутри прямоугольной области (включая границы), иначе - false.

## Format

Получает значения координат прямоугольной области в виде строки.

```
string Format(
    string & fmt,      // формат
) const
```

### Параметры

*fmt*

[in] Стока с описанием формата.

### Возвращаемое значение

Строка с координатами прямоугольной области.

## Структура CDateTime

Структура CDateTime предназначена для работы с датой и временем.

### Описание

Структура CDateTime является наследником [MqlDateTime](#), она используется для работы с датой и временем в элементах управления.

### Декларация

```
struct CDateTime
```

### Заголовок

```
#include <Tools\DateTime.mqh>
```

### Методы

Свойства	
<a href="#">MonthName</a>	Возвращает наименование месяца
<a href="#">ShortMonthName</a>	Возвращает краткое наименование месяца
<a href="#">DayName</a>	Возвращает полное наименование дня недели
<a href="#">ShortDayName</a>	Возвращает краткое наименование дня недели
<a href="#">DaysInMonth</a>	Возвращает количество дней в месяце
<b>Методы установки/получения значений</b>	
<a href="#">DateTime</a>	Возвращает/Устанавливает дату и время
<a href="#">Date</a>	Устанавливает дату
<a href="#">Time</a>	Устанавливает время
<a href="#">Sec</a>	Устанавливает количество секунд
<a href="#">Min</a>	Устанавливает количество минут
<a href="#">Hour</a>	Устанавливает час
<a href="#">Day</a>	Устанавливает день месяца
<a href="#">Mon</a>	Устанавливает месяц
<a href="#">Year</a>	Устанавливает год
<b>Дополнительные методы</b>	
<a href="#">SecDec</a>	Вычитает из даты заданное количество секунд

<a href="#"><u>SecInc</u></a>	Добавляет к дате заданное количество секунд
<a href="#"><u>MinDec</u></a>	Вычитает из даты заданное количество минут
<a href="#"><u>MinInc</u></a>	Добавляет к дате заданное количество минут
<a href="#"><u>HourDec</u></a>	Вычитает из даты заданное количество часов
<a href="#"><u>HourInc</u></a>	Добавляет к дате заданное количество часов
<a href="#"><u>DayDec</u></a>	Вычитает из даты заданное количество дней
<a href="#"><u>DayInc</u></a>	Добавляет к дате заданное количество дней
<a href="#"><u>MonDec</u></a>	Вычитает из даты заданное количество месяцев
<a href="#"><u>MonInc</u></a>	Добавляет к дате заданное количество месяцев
<a href="#"><u>YearDec</u></a>	Вычитает из даты заданное количество лет
<a href="#"><u>YearInc</u></a>	Добавляет к дате заданное количество лет

## MonthName

Возвращает наименование месяца.

```
string MonthName() const
```

Возвращает наименование месяца по номеру.

```
string MonthName(  
    const int      num          // номер месяца  
) const
```

### Параметры

*num*

[in] Номер месяца (1-12).

### Возвращаемое значение

Полное наименование месяца.

## ShortMonthName

Возвращает краткое наименование месяца.

```
string ShortMonthName() const
```

Возвращает краткое наименование месяца по номеру.

```
string ShortMonthName(  
    const int      num          // номер месяца  
) const
```

### Параметры

*num*

[in] Номер месяца (1-12).

### Возвращаемое значение

Краткое наименование месяца.

## DayName

Возвращает полное наименование дня недели.

```
string DayName() const
```

Возвращает полное наименование дня недели по номеру.

```
string DayName(  
    const int      num          // номер дня недели  
) const
```

### Параметры

*num*

[in] Номер дня недели (0-6).

### Возвращаемое значение

Полное наименование дня недели.

## ShortDayName

Возвращает краткое наименование дня недели.

```
string ShortDayName() const
```

Возвращает краткое наименование дня недели по номеру.

```
string ShortDayName(  
    const int      num           // номер дня недели  
) const
```

### Параметры

*num*

[in] Номер дня недели (0-6).

### Возвращаемое значение

Краткое наименование дня недели.

## DaysInMonth

Возвращает количество дней в месяце.

```
int DaysInMonth() const
```

### Возвращаемое значение

Количество дней в месяце.

## DateTime (Метод Get)

Возвращает дату и время.

```
datetime DateTime()
```

### Возвращаемое значение

Значение типа [datetime](#).

## DateTime (Метод Set datetime)

Устанавливает дату и время с использованием типа [datetime](#).

```
void DateTime(
    const datetime      value          // дата и время
)
```

### Параметры

*value*

[in] Значение типа [datetime](#).

### Возвращаемое значение

Нет.

## DateTime (Метод Set MqlDateTime)

Устанавливает дату и время с использованием типа [MqlDateTime](#).

```
void DateTime(
    const MqlDateTime   &value          // дата и время
)
```

### Параметры

*value*

[in] Значение типа [MqlDateTime](#).

### Возвращаемое значение

Нет.

## Date (Метод Set datetime)

Устанавливает дату с использованием типа [datetime](#).

```
void Date(
    const datetime      value          // дата
)
```

### Параметры

*value*  
[in] Значение типа [datetime](#).

### Возвращаемое значение

Нет.

## Date (Метод Set MqlDateTime)

Устанавливает дату с использованием типа [MqlDateTime](#).

```
void Date(
    const MqlDateTime   &value          // дата
)
```

### Параметры

*value*  
[in] Значение типа [MqlDateTime](#).

### Возвращаемое значение

Нет.

## Time (Метод Set datetime)

Устанавливает время с использованием типа [datetime](#).

```
void Time(
    const datetime value           // время
)
```

### Параметры

*value*  
[in] Значение типа [datetime](#).

### Возвращаемое значение

Нет.

## Time (Метод Set MqlDateTime)

Устанавливает время с использованием типа [MqlDateTime](#).

```
void Time(
    const MqlDateTime &value           // время
)
```

### Параметры

*value*  
[in] Значение типа [MqlDateTime](#).

### Возвращаемое значение

Нет.

## Sec

Устанавливает количество секунд.

```
void Sec(
    const int value           // количество секунд
)
```

### Параметры

*value*

[in] Количество секунд.

### Возвращаемое значение

Нет.

## Min

Устанавливает количество минут.

```
void Min(
    const int value           // количество минут
)
```

### Параметры

*value*

[in] Количество минут.

### Возвращаемое значение

Нет.

## Hour

Устанавливает час.

```
void Hour(  
    const int    value          // час  
)
```

### Параметры

*value*  
[in] Час.

### Возвращаемое значение

Нет.

## Day

Устанавливает день месяца.

```
void Day(
    const int value           // день месяца
)
```

### Параметры

*value*

[in] День месяца.

### Возвращаемое значение

Нет.

## Mon

Устанавливает месяц.

```
void Mon(
    const int value           // номер месяца
)
```

### Параметры

*value*  
[in] Номер месяца.

### Возвращаемое значение

Нет.

## Year

Устанавливает год.

```
void Day(
    const int value           // год
)
```

### Параметры

*value*

[in] Номер года.

### Возвращаемое значение

Нет.

## SecDec

Вычитает из даты заданное количество секунд.

```
void SecDec(
    int   delta=1           // количество секунд
)
```

### Параметры

*delta*

[in] Количество секунд.

### Возвращаемое значение

Нет.

## SecInc

Добавляет к дате заданное количество секунд.

```
void SecInc(
    int delta=1           // количество секунд
)
```

### Параметры

*delta*

[in] Количество секунд.

### Возвращаемое значение

Нет.

## MinDec

Вычитает из даты заданное количество минут.

```
void MinDec(  
    int    delta=1           // количество минут  
)
```

### Параметры

*delta*

[in] Количество минут.

### Возвращаемое значение

Нет.

## MinInc

Добавляет к дате заданное количество минут.

```
void MinInc(  
    int delta=1           // количество минут  
)
```

### Параметры

*delta*

[in] Количество минут.

### Возвращаемое значение

Нет.

## HourDec

Вычитает из даты заданное количество часов.

```
void HourDec(  
    int delta=1           // количество часов  
)
```

### Параметры

*delta*

[in] Количество часов.

### Возвращаемое значение

Нет.

## HourInc

Добавляет к дате заданное количество часов.

```
void HourInc(  
    int delta=1           // количество часов  
)
```

### Параметры

*delta*

[in] Количество часов.

### Возвращаемое значение

Нет.

## DayDec

Вычитает из даты заданное количество дней.

```
void DayDec(  
    int delta=1           // количество дней  
)
```

### Параметры

*delta*

[in] Количество дней.

### Возвращаемое значение

Нет.

## DayInc

Добавляет к дате заданное количество дней.

```
void DayInc(
    int delta=1           // days
)
```

### Параметры

*delta*

[in] Количество дней.

### Возвращаемое значение

Нет.

## MonDec

Вычитает из даты заданное количество месяцев.

```
void MonDec(
    int   delta=1           // months
)
```

### Параметры

*delta*

[in] Количество месяцев.

### Возвращаемое значение

Нет.

## MonInc

Добавляет к дате заданное количество месяцев.

```
void MonInc(
    int   delta=1           // количество месяцев
)
```

### Параметры

*delta*  
[in] Months to add.

### Возвращаемое значение

Нет.

## YearDec

Вычитает из даты заданное количество лет.

```
void YearDec(  
    int delta=1           // количество лет  
)
```

### Параметры

*delta*

[in] Количество лет.

### Возвращаемое значение

Нет.

## YearInc

Добавляет к дате заданное количество лет.

```
void YearInc(  
    int    delta=1           // количество лет  
)
```

### Параметры

*delta*

[in] Количество лет.

### Возвращаемое значение

Нет.

## Класс CWnd

Класс CWnd является базовым классом для всех элементов управления Стандартной библиотеки.

### Описание

Класс CWnd является программной реализацией базового класса элемента объекта управления.

### Декларация

```
class CWnd : public CObject
```

### Заголовок

```
#include <Controls\Wnd.mqh>
```

### Иерархия наследования

[CObject](#)

CWnd

#### Прямые потомки

[CDragWnd](#), [CWndContainer](#), [CWndObj](#)

### Методы класса по группам

Создание и уничтожение	
<a href="#">Create</a>	Создает элемент
<a href="#">Destroy</a>	Уничтожает элемент
Обработчики событий графика	
<a href="#">OnEvent</a>	Обрабатывает все события графика
<a href="#">OnMouseEvent</a>	Обрабатывает только событие графика <a href="#">CHARTEVENT_MOUSE_MOVE</a>
Наименование	
<a href="#">Name</a>	Получает имя элемента
Механизм доступа к контейнеру	
<a href="#">ControlsTotal</a>	Получает количество элементов в контейнере
<a href="#">Control</a>	Получает элемент контейнера по указанному индексу
<a href="#">ControlFind</a>	Получает элемент контейнера по указанному идентификатору
Геометрия	
<a href="#">Rect</a>	Получает координаты элемента

<u>Left</u>	Получает/устанавливает X-координату левого верхнего угла элемента
<u>Top</u>	Получает/устанавливает Y-координату левого верхнего угла элемента
<u>Right</u>	Получает/устанавливает X-координату правого нижнего угла элемента
<u>Bottom</u>	Получает/устанавливает Y-координату правого нижнего угла элемента
<u>Width</u>	Получает/устанавливает ширину элемента
<u>Height</u>	Получает/устанавливает высоту элемента
<u>Move</u>	Абсолютное перемещение элемента
<u>Shift</u>	Относительное перемещение элемента
<u>Resize</u>	Изменяет размеры элемента
<u>Contains</u>	Проверяет факт нахождения точки/элемента в области графика, занятой элементом
<b>Выравнивание</b>	
<u>Alignment</u>	Устанавливает параметры выравнивания элемента
<u>Align</u>	Выравнивает элемент в указанной области графика
<b>Идентификация</b>	
<u>Id</u>	Получает/устанавливает идентификатор элемента
<b>Состояние</b>	
<u>.IsEnabled</u>	Проверяет способность элемента отвечать на действия пользователя
<u>Enable</u>	Включает способность элемента отвечать на действия пользователя
<u>Disable</u>	Отключает способность элемента отвечать на действия пользователя
<u>IsVisible</u>	Проверяет флаг видимости элемента
<u>Visible</u>	Устанавливает флаг видимости элемента на графике
<u>Show</u>	Отображает элемент управления на графике
<u>Hide</u>	Скрывает элемент управления с графика
<u>IsActive</u>	Проверяет активность элемента

<a href="#"><u>Activate</u></a>	Делает элемент активным
<a href="#"><u>Deactivate</u></a>	Делает элемент неактивным
<b>Флаги состояния</b>	
<a href="#"><u>StateFlags</u></a>	Получает/изменяет флаги состояния элемента
<a href="#"><u>StateFlagsSet</u></a>	Устанавливает флаги состояния элемента
<a href="#"><u>StateFlagsReset</u></a>	Сбрасывает флаги состояния элемента
<b>Флаги свойств</b>	
<a href="#"><u>PropFlags</u></a>	Получает/изменяет флаги свойств элемента
<a href="#"><u>PropFlagsSet</u></a>	Устанавливает флаги свойств элемента
<a href="#"><u>PropFlagsReset</u></a>	Сбрасывает флаги свойств элемента
<b>Для операций с мышью</b>	
<a href="#"><u>MouseX</u></a>	Получает/сохраняет координату X курсора мыши
<a href="#"><u>MouseY</u></a>	Получает/сохраняет координату Y курсора мыши
<a href="#"><u>MouseFlags</u></a>	Получает/сохраняет состояние кнопок мыши
<a href="#"><u>MouseFocusKill</u></a>	Отбирает фокус мыши
<b>Обработка внутренних событий</b>	
<a href="#"><u>OnCreate</u></a>	Обработчик события "Create" элемента управления
<a href="#"><u>OnDestroy</u></a>	Обработчик события "Destroy" элемента управления
<a href="#"><u>OnMove</u></a>	Обработчик события "Move" элемента управления
<a href="#"><u>OnResize</u></a>	Обработчик события "Resize" элемента управления
<a href="#"><u>OnEnable</u></a>	Обработчик события "Enable" элемента управления
<a href="#"><u>OnDisable</u></a>	Обработчик события "Disable" элемента управления
<a href="#"><u>OnShow</u></a>	Обработчик события "Show" элемента управления
<a href="#"><u>OnHide</u></a>	Обработчик события "Hide" элемента управления

<a href="#"><u>OnActivate</u></a>	Обработчик события "Activate" элемента управления
<a href="#"><u>OnDeactivate</u></a>	Обработчик события "Deactivate" элемента управления
<a href="#"><u>OnClick</u></a>	Обработчик события "Click" элемента управления
<a href="#"><u>OnChange</u></a>	Обработчик события "Change" элемента управления
<b>Обработка событий мыши</b>	
<a href="#"><u>OnMouseDown</u></a>	Обработчик события "MouseDown" элемента управления
<a href="#"><u>OnMouseUp</u></a>	Обработчик события "MouseUp" элемента управления
<b>Обработка событий перетаскивания</b>	
<a href="#"><u>OnDragStart</u></a>	Обработчик события "DragStart" элемента управления
<a href="#"><u>OnDragProcess</u></a>	Обработчик события "DragProcess" элемента управления
<a href="#"><u>OnDragEnd</u></a>	Обработчик события "DragEnd" элемента управления
<b>Работа с объектом</b>	
<a href="#"><u>DragObjectCreate</u></a>	Создает объект перетаскивания
<a href="#"><u>DragObjectDestroy</u></a>	Уничтожает объект перетаскивания

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

## Create

Создает элемент управления.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата x1
    const int     y1,              // координата y1
    const int     x2,              // координата x2
    const int     y2               // координата y2
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Метод базового класса только сохраняет параметры создания и всегда возвращает true.

## Destroy

Уничтожает элемент управления.

```
virtual bool Destroy()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&   lparam,        // параметр события
    const double& dparam,       // параметр события
    const string& sparam        // параметр события
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMouseEvent

Обрабатывает события мыши (событие графика [CHARTEVENT\\_MOUSE\\_MOVE](#)).

```
virtual bool OnMouseEvent(
    const int   x,           // координата x
    const int   y,           // координата y
    const int   flags        // флаги
)
```

### Параметры

*x*

[in] Координата X курсора мышки относительно верхнего левого угла графика.

*y*

[in] Координата Y курсора мышки относительно верхнего левого угла графика.

*flags*

[in] Флаги состояния кнопок мыши.

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Name

Получает имя объекта управления.

```
string Name() const
```

### Возвращаемое значение

Имя объекта управления.

## ControlsTotal

Получает количество элементов управления в контейнере.

```
int ControlsTotal() const
```

### Возвращаемое значение

Количество элементов управления в контейнере.

### Примечание

Метод базового класса не имеет контейнера элементов, обеспечивая только механизм доступа для своих потомков, и всегда возвращает 0.

## Control

Получает элемент управления из контейнера по указанному индексу.

```
CWnd* Control(  
    const int ind // индекс  
) const
```

### Параметры

*ind*

[in] Индекс искомого элемента в контейнере.

### Возвращаемое значение

Указатель на элемент управления из контейнера.

### Примечание

Метод базового класса не имеет контейнера элементов, обеспечивая только механизм доступа для своих потомков, и всегда возвращает NULL.

## ControlFind

Получает элемент управления из контейнера по указанному идентификатору.

```
virtual CWnd* ControlFind(
    const long id // идентификатор
)
```

### Параметры

*id*

[in] Идентификатор искомого элемента управления.

### Возвращаемое значение

Указатель на элемент управления из контейнера.

### Примечание

Метод базового класса не имеет контейнера элементов, обеспечивая механизм доступа для своих потомков. При совпадении искомого идентификатора с собственным возвращает указатель на себя (this).

## Rect

Получает указатель на объект с координатами элемента управления.

```
const CRect* Rect() const
```

### Возвращаемое значение

Указатель на объект.

## Left (метод Get)

Получает координату X левой верхней точки элемента управления.

```
int Left()
```

### Возвращаемое значение

Координата X левой верхней точки элемента управления.

## Left (метод Set)

Устанавливает координату X левой верхней точки элемента управления.

```
virtual void Left(  
    const int x // координата  
)
```

### Параметры

x

[in] Новое значение координаты X левой верхней точки.

### Возвращаемое значение

Нет.

## Top (метод Get)

Получает координату Y левой верхней точки элемента управления.

```
int Top()
```

### Возвращаемое значение

Координата Y левой верхней точки элемента управления.

## Top (метод Set)

Устанавливает координату Y левой верхней точки элемента управления.

```
virtual void Top(  
    const int y // координата y  
)
```

### Параметры

*y*

[in] Новое значение координаты Y левой верхней точки.

### Возвращаемое значение

Нет.

## Right (метод Get)

Получает координату X правой нижней точки элемента управления.

```
int Right()
```

### Возвращаемое значение

Координата X правой нижней точки элемента управления.

## Right (метод Set)

Устанавливает координату X правой нижней точки элемента управления.

```
virtual void Right(  
    const int x // координата x  
)
```

### Параметры

x

[in] Новое значение координаты X правой нижней точки.

### Возвращаемое значение

Нет.

## Bottom (метод Get)

Получает координату Y правой нижней точки элемента управления.

```
int Bottom()
```

### Возвращаемое значение

Координата Y правой нижней точки элемента управления.

## Bottom (метод Set)

Устанавливает координату Y правой нижней точки элемента управления.

```
virtual void Bottom(  
    const int y // координата y  
)
```

### Параметры

*y*

[in] Новое значение координаты Y правой нижней точки.

### Возвращаемое значение

Нет.

## Width (метод Get)

Получает ширину элемента управления.

```
int Width()
```

### Возвращаемое значение

Ширина элемента управления.

## Width (метод Set)

Устанавливает ширину элемента управления.

```
virtual bool Width(  
    const int w // ширина  
)
```

### Параметры

w

[in] Новое значение ширины.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Height (метод Get)

Получает высоту элемента управления.

```
int Height()
```

### Возвращаемое значение

Высота элемента управления.

## Height (метод Set)

Устанавливает высоту элемента управления.

```
virtual bool Height(
    const int h           // высота
)
```

### Параметры

*h*

[in] Новое значение высоты.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Move

Осуществляет абсолютное перемещение элемента управления.

```
virtual bool Move(
    const int x,           // координата x
    const int y            // координата y
)
```

### Параметры

*x*

[in] Новое значение координаты X левой верхней точки.

*y*

[in] Новое значение координаты Y левой верхней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Shift

Осуществляет относительное перемещение элемента управления.

```
virtual bool Shift(
    const int dx,           // смещение по x
    const int dy            // смещение по y
)
```

### Параметры

*dx*

[in] Относительное смещение по координате X.

*dy*

[in] Относительное смещение по координате Y.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Resize

Изменяет размеры элемента управления.

```
virtual bool Resize(  
    const int w,  
    const int h  
)
```

### Параметры

*w*

[in] Новая ширина.

*h*

[in] Новая высота.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Contains

Проверяет факт нахождения точки в области графика, занятой элементом управления.

```
bool Contains(
    const int x,           // координата X
    const int y            // координата Y
)
```

### Параметры

*x*

[in] Координата X точки.

*y*

[in] Координата Y точки.

### Возвращаемое значение

true - если точка находится внутри области (включая границы), иначе false.

## Contains

Проверяет факт нахождения указанного элемента в области графика, занятой элементом управления.

```
bool Contains(
    const CWnd* control      // указатель
) const
```

### Параметры

*control*

[in] Указатель на элемент.

### Возвращаемое значение

true - если элемент находится внутри области (включая границы), иначе false.

## Alignment

Устанавливает параметры выравнивания элемента управления.

```
void Alignment(
    const int flags,           // флаги
    const int left,            // отступ
    const int top,             // отступ
    const int right,           // отступ
    const int bottom           // отступ
)
```

### Параметры

*flags*

[in] Флаги выравнивания элемента управления.

*left*

[in] Фиксированный отступ от левого края.

*top*

[in] Фиксированный отступ от верхнего края.

*right*

[in] Фиксированный отступ от правого края.

*bottom*

[in] Фиксированный отступ от нижнего края.

### Возвращаемое значение

Нет.

### Примечание

Флаги выравнивания:

```
enum WND_ALIGN_FLAGS
{
    WND_ALIGN_NONE=0,                      // нет выравнивания
    WND_ALIGN_LEFT=1,                      // выравнивание по левой границе
    WND_ALIGN_TOP=2,                       // выравнивание по верхней границе
    WND_ALIGN_RIGHT=4,                     // выравнивание по правой границе
    WND_ALIGN_BOTTOM=8,                    // выравнивание по нижней границе
    WND_ALIGN_WIDTH = WND_ALIGN_LEFT|WND_ALIGN_RIGHT, // выравнивание по ширине
    WND_ALIGN_HEIGHT=WND_ALIGN_TOP|WND_ALIGN_BOTTOM, // выравнивание по высоте
    WND_ALIGN_CLIENT=WND_ALIGN_WIDTH|WND_ALIGN_HEIGHT, // выравнивание по ширине и высоте
}
```

## Align

Выравнивает элемент в указанной области графика.

```
virtual bool Align(
    const CRect* rect      // указатель
)
```

### Параметры

*rect*

[in] Указатель на объект с координатами области графика.

### Возвращаемое значение

true-в случае удачи, иначе false.

### Примечание

Параметры выравнивания должны быть заданы заранее (по умолчанию выравнивания нет).

## Id (метод Get)

Получает идентификатор элемента управления.

```
long Id() const
```

### Возвращаемое значение

Значение идентификатора.

## Id (метод Set)

Устанавливает значение идентификатора элемента управления.

```
virtual long Id(  
    const long id // идентификатор  
)
```

### Параметры

x

[in] Новое значение идентификатора элемента управления.

### Возвращаемое значение

Нет.

## IsEnabled

Проверяет способность элемента управления отвечать на действия пользователя.

```
bool IsEnabled() const
```

### Возвращаемое значение

true - в случае, если элемент управления способен отвечать на действия пользователя, иначе - false.

## Enable

Включает способность элемента отвечать на действия пользователя.

```
virtual bool Enable()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

При включенном режиме элемент отрабатывает внешние события.

## Disable

Отключает способность элемента отвечать на действия пользователя.

```
virtual bool Disable()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

При отключенном режиме элемент не отрабатывает внешние события.

## IsVisible

Проверяет видимость элемента управления.

```
bool IsVisible() const
```

### Возвращаемое значение

true - в случае, если элемент управления отображается на графике, иначе - false.

## Visible

Устанавливает флаг видимости элемента управления в указанное состояние.

```
virtual bool Visible(
    const bool flag          // флаг
)
```

### Параметры

*flag*

[in] Новое состояние флага.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Show

Делает элемент управления видимым.

```
virtual bool Show()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Hide

Делает элемент управления невидимым.

```
virtual bool Hide()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## IsActive

Проверяет активность элемента управления.

```
bool IsActive() const
```

### Возвращаемое значение

true - в случае, если элемент управления активен, иначе - false.

## Activate

Делает элемент управления активным.

```
virtual bool Activate()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Элемент становится активным при наведении на него курсора мыши.

## Deactivate

Делает элемент управления неактивным.

```
virtual bool Deactivate()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Элемент становится неактивным при выведении с него курсора мыши.

## StateFlags (метод Get)

Получает флаги свойств элемента управления.

```
int StateFlags()
```

### Возвращаемое значение

Флаги состояния элемента управления.

## StateFlags (метод Set)

Устанавливает флаги состояния элемента управления.

```
virtual void StateFlags(  
    const int flags // флаги  
)
```

### Параметры

*flags*

[in] Новая комбинация флагов состояния.

### Возвращаемое значение

Нет.

## StateFlagsSet

Устанавливает флаги свойств элемента управления.

```
virtual void StateFlagsSet(
    const int flags        // флаги
)
```

### Параметры

*flags*

[in] Комбинация флагов свойств, подлежащих установке.

### Возвращаемое значение

Нет.

## StateFlagsReset

Сбрасывает флаги свойств элемента управления.

```
virtual void StateFlagsReset(
    const int flags        // флаги
)
```

### Параметры

*flags*

[in] Комбинация флагов свойств, подлежащих сбросу.

### Возвращаемое значение

Нет.

## PropFlags (метод Get)

Получает флаги свойств элемента управления.

```
void PropFlags(
    const int flags        // флаги
)
```

### Возвращаемое значение

Флаги свойств элемента управления.

## PropFlags (метод Set)

Устанавливает флаги свойств элемента управления.

```
virtual void PropFlags(
    const int flags        // флаги
)
```

### Параметры

*flags*

[in] Новая комбинация флагов свойств.

### Возвращаемое значение

Нет.

## PropFlagsSet

Устанавливает флаги свойств элемента управления.

```
virtual void PropFlagsSet(
    const int flags        // флаги
)
```

### Параметры

*flags*

[in] Комбинация флагов свойств, подлежащих установке.

### Возвращаемое значение

Нет.

## PropFlagsReset

Сбрасывает флаги свойств элемента управления.

```
virtual void PropFlagsReset(
    const int flags        // флаги
)
```

### Параметры

*flags*

[in] Комбинация флагов свойств, подлежащих сбросу.

### Возвращаемое значение

Нет.

## MouseX (метод Set)

Сохраняет координату X курсора мыши.

```
void MouseX(
    const int value      // координата
)
```

### Параметры

*value*

[in] Координата X курсора мыши.

### Возвращаемое значение

Нет.

## MouseX (метод Get)

Получает сохраненную координату X курсора мыши.

```
int MouseX()
```

### Возвращаемое значение

Координата X курсора мыши.

## MouseY (метод Set)

Сохраняет координату Y курсора мыши.

```
void MouseY(
    const int value      // координата
)
```

### Параметры

*value*

[in] Координата Y курсора мыши.

### Возвращаемое значение

Нет.

## MouseY (метод Get)

Получает сохраненную координату Y курсора мыши.

```
int MouseY()
```

### Возвращаемое значение

Координата Y курсора мыши.

## MouseFlags (метод Set)

Сохраняет состояние кнопок мыши.

```
virtual void MouseFlags(
    const int value      // состояние
)
```

### Параметры

*value*

[in] Состояние кнопок мыши.

### Возвращаемое значение

Нет.

## MouseFlags (метод Get)

Получает сохраненное состояние кнопок мыши.

```
int MouseFlags()
```

### Возвращаемое значение

Состояние кнопок мыши.

## MouseFocusKill

Очищает сохраненное состояние мыши и деактивирует элемент управления.

```
bool MouseFocusKill(
    const long id=CONTROLS_INVALID_ID           // идентификатор
)
```

### Параметры

*id=CONTROLS\_INVALID\_ID*

[in] Идентификатор элемента управления, получившего фокус мыши.

### Возвращаемое значение

Результат выполнения деактивации элемента управления.

## OnCreate

Виртуальный обработчик внутреннего события "Create" (создание) элемента управления.

```
virtual bool OnCreate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnDestroy

Виртуальный обработчик внутреннего события "Destroy" (уничтожение) элемента управления.

```
virtual bool OnDestroy()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnMove

Виртуальный обработчик внутреннего события "Move" (перемещение) элемента управления.

```
virtual bool OnMove()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnResize

Виртуальный обработчик внутреннего события "Resize" (изменение размеров) элемента управления.

```
virtual bool OnResize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnEnable

Виртуальный обработчик внутреннего события "Enable" (включение способности отвечать на действия пользователя) элемента управления.

```
virtual bool OnEnable()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnDisable

Виртуальный обработчик внутреннего события "Disable" (отключение способности отвечать на действия пользователя) элемента управления.

```
virtual bool OnDisable()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnShow

Виртуальный обработчик внутреннего события "Show" (отображение) элемента управления.

```
virtual bool OnShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) элемента управления.

```
virtual bool OnHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnActivate

Виртуальный обработчик внутреннего события "Activate" (активизация) элемента управления.

```
virtual bool OnActivate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnDeactivate

Виртуальный обработчик внутреннего события "Deactivate" (деактивизация) элемента управления.

```
virtual bool OnDeactivate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnClick

Виртуальный обработчик внутреннего события "Click" (клик мыши) элемента управления.

```
virtual bool OnClick()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChange

Виртуальный обработчик внутреннего события "Change" (изменение) элемента управления.

```
virtual bool OnChange()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnMouseDown

Виртуальный обработчик события мыши "MouseDown" (нажатие кнопки мыши) элемента управления.

```
virtual bool OnMouseDown()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "MouseDown" происходит при нажатии левой кнопки мыши, если указатель мыши находится на элементе управления.

## OnMouseUp

Виртуальный обработчик события мыши "MouseUp" (отпускание кнопки мыши) элемента управления.

```
virtual bool OnMouseUp()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "MouseUp" происходит при отпускании левой кнопки мыши, если указатель мыши находится на элементе управления.

## OnDragStart

Виртуальный обработчик события "DragStart" (начало операции перетаскивания) элемента управления.

```
virtual bool OnDragStart()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "DragStart" происходит при начале операции перетаскивания элемента управления.

## OnDragProcess

Виртуальный обработчик события "DragProcess" (операция перетаскивания) элемента управления.

```
virtual bool OnDragProcess(
    const int x,           // координата x
    const int y            // координата y
)
```

### Параметры

*x*

[in] Текущее значение координаты X курсора мыши.

*y*

[in] Текущее значение координаты Y курсора мыши.

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "DragProcess" происходит при перемещении элемента управления.

## OnDragEnd

Виртуальный обработчик события "DragEnd" (завершение операции перетаскивания) элемента управления.

```
virtual bool OnDragEnd()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "DragEnd" происходит по завершении операции перетаскивания элемента управления.

## DragObjectCreate

Создает объект перетаскивания.

```
virtual bool DragObjectCreate()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## DragObjectDestroy

Уничтожает объект перетаскивания.

```
virtual bool DragObjectDestroy()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Класс CWndObj

Класс CWndObj является базовым классом для простых (основанных на объектах графика) элементов управления Стандартной библиотеки.

### Описание

Класс CWndObj является программной реализацией основы простого элемента управления.

### Декларация

```
class CWndObj : public CWnd
```

### Заголовок

```
#include <Controls\WndObj.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndObj
```

### Прямые потомки

[CBmpButton](#), [CButton](#), [CEdit](#), [CLabel](#), [CPanel](#), [CPicture](#)

### Методы класса по группам

Обработчики событий графика	
<a href="#">OnEvent</a>	Обрабатывает все события графика
<b>Параметры</b>	
<a href="#">Text</a>	Получает/устанавливает значение свойства <a href="#">OBJPROP_TEXT</a> объекта графика
<a href="#">Color</a>	Получает/устанавливает значение свойства <a href="#">OBJPROP_COLOR</a> объекта графика
<a href="#">ColorBackground</a>	Получает/устанавливает значение свойства <a href="#">OBJPROP_BGCOLOR</a> объекта графика
<a href="#">ColorBorder</a>	Получает/устанавливает значение свойства <a href="#">OBJPROP_BORDER_COLOR</a> объекта графика
<a href="#">Font</a>	Получает/устанавливает значение свойства <a href="#">OBJPROP_FONT</a> объекта графика
<a href="#">FontSize</a>	Получает/устанавливает значение свойства <a href="#">OBJPROP_FONTSIZE</a> объекта графика
<a href="#">ZOrder</a>	Получает/устанавливает значение свойства <a href="#">OBJPROP_ZORDER</a> объекта графика

<b>Обработка событий объектов графика</b>	
<a href="#">OnObjectCreate</a>	Обработчик события <a href="#">CHARTEVENT_OBJECT_CREATE</a> объекта графика
<a href="#">OnObjectChange</a>	Обработчик события <a href="#">CHARTEVENT_OBJECT_CHANGE</a> объекта графика
<a href="#">OnObjectDelete</a>	Обработчик события <a href="#">CHARTEVENT_OBJECT_DELETE</a> объекта графика
<a href="#">OnObjectDrag</a>	Обработчик события <a href="#">CHARTEVENT_OBJECT_DRAG</a> объекта графика
<b>Обработка изменения параметров</b>	
<a href="#">OnSetText</a>	Обработчик события "SetText" элемента управления
<a href="#">OnSetColor</a>	Обработчик события "SetColor" элемента управления
<a href="#">OnSetColorBackground</a>	Обработчик события "SetColorBackground" элемента управления
<a href="#">OnSetFont</a>	Обработчик события "SetFont" элемента управления
<a href="#">OnSetFontSize</a>	Обработчик события "SetFontSize" элемента управления
<a href="#">OnSetZOrder</a>	Обработчик события "SetZOrder" элемента управления
<b>Обработка внутренних событий</b>	
<a href="#">OnDestroy</a>	Обработчик внутреннего события "Destroy" элемента управления
<a href="#">OnChange</a>	Обработчик внутреннего события "Change" элемента управления

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

**Методы унаследованные от CWnd**

[Create](#), [Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)



## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Text (метод Get)

Получает значение свойства [OBJPROP\\_TEXT](#) (текст) объекта графика.

```
string Text()
```

### Возвращаемое значение

Значение свойства [OBJPROP\\_TEXT](#) объекта графика.

## Text (метод Set)

Устанавливает значение свойства [OBJPROP\\_TEXT](#) (текст) объекта графика.

```
bool Text(  
    const string value // значение  
)
```

### Параметры

*value*

[in] Новое значение свойства [OBJPROP\\_TEXT](#).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Color (метод Get)

Получает значение свойства [OBJPROP\\_COLOR](#) (цвет) объекта графика.

```
color Color()
```

### Возвращаемое значение

Значение свойства [OBJPROP\\_COLOR](#) объекта графика.

## Color (метод Set)

Устанавливает значение свойства [OBJPROP\\_COLOR](#) (цвет) объекта графика.

```
bool Color(  
    const color value // значение  
)
```

### Параметры

*value*

[in] Новое значение свойства [OBJPROP\\_COLOR](#).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ColorBackground (метод Get)

Получает значение свойства [OBJPROP\\_BGCOLOR](#) (цвет фона) объекта графика.

```
color ColorBackground()
```

### Возвращаемое значение

Значение свойства [OBJPROP\\_BGCOLOR](#) объекта графика.

## ColorBackground (метод Set)

Устанавливает значение свойства [OBJPROP\\_BGCOLOR](#) (цвет фона) объекта графика.

```
bool ColorBackground(  
    const color value // значение  
)
```

### Параметры

*value*

[in] Новое значение свойства [OBJPROP\\_BGCOLOR](#).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ColorBorder (метод Get)

Получает значение свойства [OBJPROP\\_BORDER\\_COLOR](#) (цвет рамки) объекта графика.

```
color ColorBorder()
```

### Возвращаемое значение

Значение свойства [OBJPROP\\_BORDER\\_COLOR](#) объекта графика.

## ColorBorder (метод Set)

Устанавливает значение свойства [OBJPROP\\_BORDER\\_COLOR](#) (цвет рамки) объекта графика.

```
bool ColorBorder(  
    const color value // значение  
)
```

### Параметры

*value*

[in] Новое значение свойства [OBJPROP\\_BORDER\\_COLOR](#).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Font (метод Get)

Получает значение свойства [OBJPROP\\_FONT](#) (шрифт) объекта графика.

```
string Font()
```

### Возвращаемое значение

Значение свойства [OBJPROP\\_FONT](#) объекта графика.

## Font (метод Set)

Устанавливает значение свойства [OBJPROP\\_FONT](#) (шрифт) объекта графика.

```
bool Font(  
    const string value // значение  
)
```

### Параметры

*value*

[in] Новое значение свойства [OBJPROP\\_FONT](#).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## FontSize (метод Get)

Получает значение свойства [OBJPROP\\_FONTSIZE](#) (размер шрифта) объекта графика.

```
int FontSize()
```

### Возвращаемое значение

Значение свойства [OBJPROP\\_FONTSIZE](#) объекта графика.

## FontSize (метод Set)

Устанавливает значение свойства [OBJPROP\\_FONTSIZE](#) (размер шрифта) объекта графика.

```
bool FontSize(  
    const int value // значение  
)
```

### Параметры

*value*

[in] Новое значение свойства [OBJPROP\\_FONTSIZE](#).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ZOrder (метод Get)

Получает значение свойства [OBJPROP\\_ZORDER](#) объекта графика.

```
long ZOrder()
```

### Возвращаемое значение

Значение свойства [OBJPROP\\_ZORDER](#) объекта графика.

## ZOrder (метод Set)

Устанавливает значение свойства [OBJPROP\\_ZORDER](#) объекта графика.

```
bool ZOrder(  
    const long value // значение  
)
```

### Параметры

*value*

[in] Новое значение свойства [OBJPROP\\_ZORDER](#).

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnObjectCreate

Виртуальный обработчик события [CHARTEVENT\\_OBJECT\\_CREATE](#) объекта графика.

```
virtual bool OnObjectCreate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnObjectChange

Виртуальный обработчик события [CHARTEVENT\\_OBJECT\\_CHANGE](#) объекта графика.

```
virtual bool OnObjectChange()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnObjectDelete

Виртуальный обработчик события [CHARTEVENT\\_OBJECT\\_DELETE](#) объекта графика.

```
virtual bool OnObjectDelete()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnObjectDrag

Виртуальный обработчик события [CHARTEVENT\\_OBJECT\\_DRAG](#) объекта графика.

```
virtual bool OnObjectDrag()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetText

Виртуальный обработчик события "SetText" (изменение свойства [OBJPROP\\_TEXT](#)) элемента управления.

```
virtual bool OnSetText()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnSetColor

Виртуальный обработчик события "SetColor" (изменение свойства [OBJPROP\\_COLOR](#)) элемента управления.

```
virtual bool OnSetColor()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnSetColorBackground

Виртуальный обработчик события "SetColorBackground" (изменение свойства [OBJPROP\\_BGCOLOR](#)) элемента управления.

```
virtual bool OnSetColorBackground()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnSetFont

Виртуальный обработчик события "SetFont" (изменение свойства [OBJPROP\\_FONT](#)) элемента управления.

```
virtual bool OnSetFont()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnSetFontSize

Виртуальный обработчик события "SetFontSize" (изменение свойства [OBJPROP\\_FONTSIZE](#)) элемента управления.

```
virtual bool OnSetFontSize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnSetZOrder

Виртуальный обработчик события "SetZOrder" (изменение параметра [OBJPROP\\_ZORDER](#)) элемента управления.

```
virtual bool OnSetZOrder()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnDestroy

Виртуальный обработчик внутреннего события "Destroy" (уничтожение) элемента управления.

```
virtual bool OnDestroy()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChange

Виртуальный обработчик внутреннего события "Change" (изменение) элемента управления.

```
virtual bool OnChange()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Класс CWndContainer

Класс CWndContainer является базовым классом для группы элементов управления Стандартной библиотеки.

### Описание

Класс CWndContainer является программной реализацией группы функционально связанных элементов управления.

### Декларация

```
class CWndContainer : public CWnd
```

### Заголовок

```
#include <Controls\WndContainer.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndContainer
```

### Прямые потомки

```
CCheckBox, CComboBox, CDateDropList, CDatePicker, CDialog, CRadioButton, CSscroll, CSpinEdit,  
CWndClient
```

### Методы класса по группам

Уничтожение	
<u>Destroy</u>	Уничтожает группу элементов
Обработчики событий графика	
<u>OnEvent</u>	Обрабатывает все события графика
<u>OnMouseEvent</u>	Обрабатывает только событие графика <u>CHARTEVENT_MOUSE_MOVE</u>
Доступ к содержимому контейнера	
<u>ControlsTotal</u>	Получает количество элементов в контейнере
<u>Control</u>	Элемент контейнера по указанному индексу
<u>ControlFind</u>	Получает элемент контейнера по указанному идентификатору
Наполнение	
<u>Add</u>	Добавляет элемент в группу (контейнер)
<u>Delete</u>	Удаляет элемент из группы (контейнера)

<b>Геометрия</b>	
<a href="#">Move</a>	Осуществляет абсолютное перемещение группы элементов
<a href="#">Shift</a>	Осуществляет относительное перемещение группы элементов
<b>Идентификация</b>	
<a href="#">Id</a>	Устанавливает идентификаторы элементов группы
<b>Состояние</b>	
<a href="#">Enable</a>	Включает способность группы элементов отвечать на действия пользователя
<a href="#">Disable</a>	Отключает способность группы элементов отвечать на действия пользователя
<a href="#">Show</a>	Делает группу элементов видимой
<a href="#">Hide</a>	Делает группу элементов невидимой
<b>Для операций с мышью</b>	
<a href="#">MouseFocusKill</a>	Отбирает фокус мыши
<b>Работа с файлами</b>	
<a href="#">Save</a>	Запись информации группы в файл
<a href="#">Load</a>	Чтение информации группы из файла
<b>Обработка внутренних событий</b>	
<a href="#">OnResize</a>	Обработчик события "Resize" группы элементов
<a href="#">OnActivate</a>	Обработчик события "Activate" группы элементов
<a href="#">OnDeactivate</a>	Обработчик события "Deactivate" группы элементов

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)**Методы унаследованные от CWnd**

[Create](#), [Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)



## Destroy

Уничтожает группу элементов управления.

```
virtual bool Destroy()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMouseEvent

Обрабатывает события мыши.

```
virtual bool OnMouseEvent(
    const int   x,           // координата
    const int   y,           // координата
    const int   flags        // флаги
)
```

### Параметры

*x*

[in] Координата X курсора мышки относительно верхнего левого угла графика.

*y*

[in] Координата Y курсора мышки относительно верхнего левого угла графика.

*flags*

[in] Флаги состояния кнопок мыши.

### Возвращаемое значение

true - если событие обработано, иначе - false.

## ControlsTotal

Получает количество элементов управления в контейнере.

```
int ControlsTotal() const
```

### Возвращаемое значение

Количество элементов управления в контейнере.

## Control

Получает элемент управления контейнера по указанному индексу.

```
CWnd* Control(
    const int ind      // индекс
) const
```

### Параметры

*ind*

[in] Индекс искомого элемента в контейнере.

### Возвращаемое значение

Указатель на элемент управления, либо NULL, если элемент не найден.

## ControlFind

Получает элемент управления контейнера по указанному идентификатору.

```
virtual CWnd* ControlFind(
    const long id           // идентификатор
)
```

### Параметры

*id*

[in] Идентификатор искомого элемента управления.

### Возвращаемое значение

Указатель на элемент управления, либо NULL, если элемент не найден.

## Add

Добавляет элемент в группу (контейнер).

```
bool Add(  
    CWnd& control          // ссылка  
)
```

### Параметры

*control*

[in] Ссылка на добавляемый элемент управления.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Delete

Удаляет элемент из группы (контейнера).

```
bool Delete(
    CWnd& control          // ссылка
)
```

### Параметры

*control*

[in] Ссылка на удаляемый элемент управления.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Move

Осуществляет абсолютное перемещение группы элементов.

```
virtual bool Move(
    const int x,           // координата X
    const int y            // координата Y
)
```

### Параметры

*x*

[in] Новое значение координаты X левой верхней точки.

*y*

[in] Новое значение координаты Y левой верхней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Shift

Осуществляет относительное перемещение группы элементов.

```
virtual bool Shift(
    const int dx,           // смещение по X
    const int dy            // смещение по Y
)
```

### Параметры

*dx*

[in] Относительное смещение по координате X.

*dy*

[in] Относительное смещение по координате Y.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Id

Раздает идентификаторы элементам группы.

```
virtual long Id(
    const long id           // идентификатор
)
```

### Параметры

*id*

[in] Базовый идентификатор группы.

### Возвращаемое значение

Количество идентификаторов, использованных группой элементов.

## Enable

Включает способность группы элементов отвечать на действия пользователя.

```
virtual bool Enable()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Disable

Отключает способность группы элементов отвечать на действия пользователя.

```
virtual bool Disable()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Show

Делает группу элементов видимой.

```
virtual bool Show()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Hide

Делает группу элементов невидимой.

```
virtual bool Hide()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## MouseFocusKill

Очищает сохраненное состояние мыши и деактивирует группу элементов.

```
virtual bool MouseFocusKill(
    const long id=CONTROLS_INVALID_ID           // идентификатор
)
```

### Параметры

*id=CONTROLS\_INVALID\_ID*

[in] Идентификатор элемента управления, получившего фокус мыши.

### Возвращаемое значение

Результат выполнения деактивации группы элементов.

## Save

Записывает информацию группы в файл.

```
virtual bool Save(
    const int file_handle // хендл
)
```

### Параметры

*file\_handle*

[in] Хендл бинарного файла, ранее открытого для записи.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Load

Читает информацию группы из файла.

```
virtual bool Load(
    const int file_handle // хендл
)
```

### Параметры

*file\_handle*

[in] Хендл бинарного файла, ранее открытого для чтения.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnResize

Виртуальный обработчик внутреннего события "Resize" (изменение размеров) элемента управления.

```
virtual bool OnResize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnActivate

Виртуальный обработчик внутреннего события "Activate" (активизация) элемента управления.

```
virtual bool OnActivate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnDeactivate

Виртуальный обработчик внутреннего события "Deactivate" (деактивизация) элемента управления.

```
virtual bool OnDeactivate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## Класс CLabel

Класс CLabel является классом простого элемента управления на основе объекта "Текстовая метка".

### Описание

Класс CLabel предназначен для создания простейших нередактируемых текстовых полей.

### Декларация

```
class CLabel : public CWndObj
```

### Заголовок

```
#include <Controls\Label.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndObj  
CLabel
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

Создание	
----------	--

<a href="#">Create</a>	Создает элемент управления
<b>Обработка изменения параметров</b>	
<a href="#">OnSetText</a>	Обработчик события "SetText" элемента управления
<a href="#">OnSetColor</a>	Обработчик события "SetColor" элемента управления
<a href="#">OnSetFont</a>	Обработчик события "SetFont" элемента управления
<a href="#">OnSetFontSize</a>	Обработчик события "SetFontSize" элемента управления
<b>Обработка внутренних событий</b>	
<a href="#">OnCreate</a>	Обработчик внутреннего события "Create" элемента управления
<a href="#">OnShow</a>	Обработчик внутреннего события "Show" элемента управления
<a href="#">OnHide</a>	Обработчик внутреннего события "Hide" элемента управления
<a href="#">OnMove</a>	Обработчик внутреннего события "Move" элемента управления

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CWnd

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

#### Методы унаследованные от CWndObj

[OnEvent](#), [Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

#### Пример создания панели с текстовой меткой:

```
//+-----+
//|                                         ControlsLabel.mq5 |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
```

```

#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CLabel"
#include <Controls\Dialog.mqh>
#include <Controls\Label.mqh>
//+-----+
//| defines
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)      // indent from left (with allowance)
#define INDENT_TOP           (11)      // indent from top (with allowance)
#define INDENT_RIGHT          (11)      // indent from right (with allowance)
#define INDENT_BOTTOM          (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)     // size by X coordinate
#define BUTTON_HEIGHT          (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)     // size by X coordinate
#define LIST_HEIGHT             (179)     // size by Y coordinate
#define RADIO_HEIGHT            (56)      // size by Y coordinate
#define CHECK_HEIGHT            (93)      // size by Y coordinate
//+-----+
//| Class CControlsDialog
//| Usage: main dialog of the Controls application
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CLabel                 m_label;           // CLabel object
public:
    CControlsDialog(void);
    ~CControlsDialog(void);

//--- create
    virtual bool      Create(const long chart,const string name,const int subwin,const
//--- chart event handler
    virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,const
protected:
    //--- create dependent controls
    bool               CreateLabel(void);
    //--- handlers of the dependent controls events
    void               OnClickLabel(void);
};

//+-----+
//| Event Handling
//+-----+

```

```
EVENT_MAP_BEGIN(CControlsDialog)

EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}

//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateLabel())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "CLabel" |
//+-----+
bool CControlsDialog::CreateLabel(void)
{
//--- coordinates
    int x1=INDENT_RIGHT;
    int y1=INDENT_TOP+CONTROLS_GAP_Y;
    int x2=x1+100;
    int y2=y1+20;
//--- create
    if(!m_label.Create(m_chart_id,m_name+"Label",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!m_label.Text("Label"))
        return(false);
    if(!Add(m_label))
        return(false);
//--- succeed
    return(true);
}
//+-----+
```

```
//| Global Variables
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- create application dialog
if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
    return(INIT_FAILED);
//--- run application
ExtDialog.Run();
//--- succeed
return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//---
Comment("");
//--- destroy dialog
ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function
//+-----+
void OnChartEvent(const int id,           // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## Create

Создает элемент управления CLabel.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата x1
    const int     y1,              // координата y1
    const int     x2,              // координата x2
    const int     y2               // координата y2
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnSetText

Виртуальный обработчик события "SetText" (изменение свойства [OBJPROP\\_TEXT](#)) элемента управления CLabel.

```
virtual bool OnSetText()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetColor

Виртуальный обработчик события "SetColor" (изменение свойства [OBJPROP\\_COLOR](#)) элемента управления CLabel.

```
virtual bool OnSetColor()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetFont

Виртуальный обработчик события "SetFont" (изменение свойства [OBJPROP\\_FONT](#)) элемента управления CLabel.

```
virtual bool OnSetFont()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetFontSize

Виртуальный обработчик события "SetFontSize" (изменение свойства [OBJPROP\\_FONTSIZE](#)) элемента управления CLabel.

```
virtual bool OnSetFontSize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnCreate

Виртуальный обработчик внутреннего события "Create" (создание) элемента управления CLabel.

```
virtual bool OnCreate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnShow

Виртуальный обработчик внутреннего события "Show" (отображение) элемента управления CLabel.

```
virtual bool OnShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) элемента управления CLabel.

```
virtual bool OnHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMove

Виртуальный обработчик внутреннего события "Move" (перемещение) элемента управления CLabel.

```
virtual bool OnMove()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Класс CBmpButton

Класс CBmpButton является классом простого элемента управления на основе объекта "Графическая метка".

### Описание

Класс CBmpButton предназначен для создания кнопки с графическим изображением.

### Декларация

```
class CBmpButton : public CWndObj
```

### Заголовок

```
#include <Controls\BmpButton.mqh>
```

### Иерархия наследования

```
CObject  
  CWnd  
    CWndObj  
      CBmpButton
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

Создание	
----------	--

<a href="#"><u>Create</u></a>	Создает элемент управления
<a href="#"><u>Параметры элемента управления</u></a>	
<a href="#"><u>Border</u></a>	Получает/устанавливает свойство "Border" элемента управления
<a href="#"><u>BmpNames</u></a>	Устанавливает имена bmp-файлов для отображения элемента управления
<a href="#"><u>BmpOffName</u></a>	Получает/устанавливает имя bmp-файла отображения элемента в состоянии OFF
<a href="#"><u>BmpOnName</u></a>	Получает/устанавливает имя bmp-файла отображения элемента в состоянии ON
<a href="#"><u>BmpPassiveName</u></a>	Получает/устанавливает имя bmp-файла для отображения элемента в неактивном состоянии
<a href="#"><u>BmpActiveName</u></a>	Получает/устанавливает имя bmp-файла для отображения элемента в активном состоянии
<a href="#"><u>Свойства</u></a>	
<a href="#"><u>Pressed</u></a>	Получает/устанавливает состояние элемента управления
<a href="#"><u>Locking</u></a>	Получает/устанавливает значение свойства "Locking" элемента управления
<a href="#"><u>Обработка внутренних событий</u></a>	
<a href="#"><u>OnSetZOrder</u></a>	Обработчик внутреннего события "SetZOrder" элемента управления
<a href="#"><u>OnCreate</u></a>	Обработчик внутреннего события "Create" элемента управления
<a href="#"><u>OnShow</u></a>	Обработчик внутреннего события "Show" элемента управления
<a href="#"><u>OnHide</u></a>	Обработчик внутреннего события "Hide" элемента управления
<a href="#"><u>OnMove</u></a>	Обработчик внутреннего события "Move" элемента управления
<a href="#"><u>OnChange</u></a>	Обработчик внутреннего события "Change" элемента управления
<a href="#"><u>OnActivate</u></a>	Обработчик внутреннего события "Activate" элемента управления
<a href="#"><u>OnDeactivate</u></a>	Обработчик внутреннего события "Deactivate" элемента управления

<a href="#">OnMouseDown</a>	Обработчик внутреннего события "MouseDown" элемента управления
<a href="#">OnMouseUp</a>	Обработчик внутреннего события "MouseUp" элемента управления

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

**Методы унаследованные от CWnd**

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

**Методы унаследованные от CWndObj**

[OnEvent](#), [Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

**Пример создания панели с графической меткой**

```

//+-----+
//|                                         ControlsBmpButton.mq5 |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CBmpButton"
#include <Controls\Dialog.mqh>
#include <Controls\BmpButton.mqh>
//+-----+
//| defines |
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)    // indent from left (with allowance)
#define INDENT_TOP           (11)    // indent from top (with allowance)
#define INDENT_RIGHT          (11)    // indent from right (with allowance)
#define INDENT_BOTTOM          (11)    // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)     // gap by X coordinate
#define CONTROLS_GAP_Y        (5)     // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)   // size by X coordinate
#define BUTTON_HEIGHT          (20)    // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)    // size by Y coordinate

```

```

//--- for group controls
#define GROUP_WIDTH (150) // size by X coordinate
#define LIST_HEIGHT (179) // size by Y coordinate
#define RADIO_HEIGHT (56) // size by Y coordinate
#define CHECK_HEIGHT (93) // size by Y coordinate
//+-----+
//| Class CControlsDialog |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CBmpButton     m_bmpbutton1;           // CBmpButton object
    CBmpButton     m_bmpbutton2;           // CBmpButton object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool    Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool    OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool           CreateBmpButton1(void);
    bool           CreateBmpButton2(void);
    //--- handlers of the dependent controls events
    void           OnClickBmpButton1(void);
    void           OnClickBmpButton2(void);
};

//+-----+
//| Event Handling |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_bmpbutton1,OnClickBmpButton1)
ON_EVENT(ON_CLICK,m_bmpbutton2,OnClickBmpButton2)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

```

```

    }

//+-----+
//| Create
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateBmpButton1())
        return(false);
    if(!CreateBmpButton2())
        return(false);
//--- succeed
    return(true);
}

//+-----+
//| Create the "BmpButton1" button
//+-----+
bool CControlsDialog::CreateBmpButton1(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+BUTTON_WIDTH;
    int y2=y1+BUTTON_HEIGHT;
//--- create
    if(!m_bmpbutton1.Create(m_chart_id,m_name+"BmpButton1",m_subwin,x1,y1,x2,y2))
        return(false);
//--- sets the name of bmp files of the control CBmpButton
    m_bmpbutton1.BmpNames("\\\\Images\\\\euro.bmp","\\\\Images\\\\dollar.bmp");
    if(!Add(m_bmpbutton1))
        return(false);
//--- succeed
    return(true);
}

//+-----+
//| Create the "BmpButton2" fixed button
//+-----+
bool CControlsDialog::CreateBmpButton2(void)
{
//--- coordinates
    int x1=INDENT_LEFT+2*(BUTTON_WIDTH+CONTROLS_GAP_X);
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+BUTTON_WIDTH;
    int y2=y1+BUTTON_HEIGHT;
//--- create
    if(!m_bmpbutton2.Create(m_chart_id,m_name+"BmpButton2",m_subwin,x1,y1,x2,y2))
        return(false);
}

```

```
//--- sets the name of bmp files of the control CBmpButton
m_bmpbutton2.BmpNames("\\Images\\euro.bmp", "\\Images\\dollar.bmp");
if(!Add(m_bmpbutton2))
    return(false);
m_bmpbutton2.Locking(true);
//--- succeed
return(true);
}

//+-----+
//| Event handler |
//+-----+

void CControlsDialog::OnClickBmpButton1(void)
{
    Comment(__FUNCTION__);
}

//+-----+
//| Event handler |
//+-----+

void CControlsDialog::OnClickBmpButton2(void)
{
    if(m_bmpbutton2.Pressed())
        Comment(__FUNCTION__+" State of the control is: On");
    else
        Comment(__FUNCTION__+" State of the control is: Off");
}

//+-----+
//| Global Variables |
//+-----+

CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function |
//+-----+

int OnInit()
{
    //--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    //--- run application
    ExtDialog.Run();
    //--- succeed
    return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function |
//+-----+

void OnDeinit(const int reason)
{
    //---
    Comment("");
}
```

```
//--- destroy dialog
ExtDialog.Destroy(reason);
}

//+-----+
//| Expert chart event function
//+-----+
void OnChartEvent(const int id,           // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## Create

Создает элемент управления CBmpButton.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата x1
    const int     y1,              // координата y1
    const int     x2,              // координата x2
    const int     y2               // координата y2
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Border (метод Get)

Получает значение параметра "Border" (ширина рамки) элемента управления CBmpButton.

```
int Border() const
```

### Возвращаемое значение

Значение параметра "Border".

## Border (метод Set)

Устанавливает значение параметра "Border" (ширина рамки) элемента управления CBmpButton.

```
bool Border(  
    const int value // значение  
)
```

### Параметры

*value*

[in] Новое значение параметра "Border".

### Возвращаемое значение

true - в случае удачи, иначе - false.

## BmpNames

Устанавливает имена bmp-файлов для отображения элемента управления CBitmapButton.

```
bool BmpNames(
    const string off="",
    const string on=""      // имя файла
)
```

### Параметры

*off*=""

[in] Имя bmp-файла для отображения элемента в состоянии OFF.

*on*=""

[in] Имя bmp-файла для отображения элемента в состоянии ON.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## BmpOffName (метод Get)

Получает имя bmp-файла отображения элемента в состоянии OFF.

```
string BmpOffName() const
```

### Возвращаемое значение

Имя bmp-файла отображения элемента в состоянии OFF.

## BmpOffName (метод Set)

Устанавливает имя bmp-файла для отображения элемента в состоянии OFF.

```
bool BmpOffName(  
    const string name // имя файла  
)
```

### Параметры

*name*

[in] Имя bmp-файла для отображения элемента в состоянии OFF.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## BmpOnName (метод Get)

Получает имя bmp-файла отображения элемента в состоянии ON.

```
string BmpOnName() const
```

### Возвращаемое значение

Имя bmp-файла отображения элемента в состоянии ON.

## BmpOnName (метод Set)

Устанавливает имя bmp-файла для отображения элемента в состоянии ON.

```
bool BmpOnName(  
    const string name // имя файла  
)
```

### Параметры

*name*

[in] Имя bmp-файла для отображения элемента в состоянии ON.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## BmpPassiveName (метод Get)

Получает имя bmp-файла для отображения элемента управления CBmpButton в неактивном состоянии.

```
string BmpPassiveName() const
```

### Возвращаемое значение

Имя bmp-файла для отображения элемента в неактивном состоянии.

## BmpPassiveName (метод Set)

Устанавливает имя bmp-файла для отображения элемента управления CBmpButton в неактивном состоянии.

```
bool BmpPassiveName(  
    const string name // имя файла  
)
```

### Параметры

*name*

[in] Имя bmp-файла для отображения элемента в неактивном состоянии.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## BmpActiveName (метод Get)

Получает имя bmp-файла для отображения элемента управления CVmrButton в активном состоянии.

```
string BmpActiveName() const
```

### Возвращаемое значение

Имя bmp-файла для отображения элемента в активном состоянии.

### Примечание

Элемент управления является активным, если указатель мыши находится на элементе управления.

## BmpActiveName (метод Set)

Устанавливает имя bmp-файла для отображения элемента CVmrButton в активном состоянии.

```
bool BmpActiveName(
    const string name // имя файла
)
```

### Параметры

*name*

[in] Имя bmp-файла для отображения элемента в активном состоянии.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Pressed (метод Get)

Получает состояние (свойство Pressed) элемента управления CBmpButton.

```
bool Pressed() const
```

### Возвращаемое значение

Состояние элемента.

## Pressed (метод Set)

Устанавливает состояние (свойство Pressed) элемента управления CBmpButton.

```
bool Pressed(  
    const bool pressed // состояние  
)
```

### Параметры

*pressed*

[in] Новое состояние элемента.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Locking (метод Get)

Получает значение свойства "Locking" (фиксация) элемента управления CBitmapButton.

```
bool Locking() const
```

### Возвращаемое значение

Значение свойства "Locking".

## Locking (метод Set)

Устанавливает свойство "Locking" (фиксация) элемента управления CBitmapButton.

```
void Locking(  
    const bool locking // значение  
)
```

### Параметры

*locking*

[in] Новое значение свойства "Locking".

### Возвращаемое значение

Нет.

## OnSetZOrder

Виртуальный обработчик события "SetZOrder" (изменение параметра [OBJPROP\\_ZORDER](#)) элемента управления CBmpButton.

```
virtual bool OnSetZOrder()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnCreate

Виртуальный обработчик внутреннего события "Create" (создание) элемента управления CBmpButton.

```
virtual bool OnCreate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnShow

Виртуальный обработчик внутреннего события "Show" (отображение) элемента управления CBmpButton.

```
virtual bool OnShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) элемента управления CButton.

```
virtual bool OnHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMove

Виртуальный обработчик внутреннего события "Move" (перемещение) элемента управления CBmpButton.

```
virtual bool OnMove()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChange

Виртуальный обработчик внутреннего события "Change" (изменение) элемента управления CBmpButton.

```
virtual bool OnChange()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnActivate

Виртуальный обработчик внутреннего события "Activate" (активизация) элемента управления CBmpButton.

```
virtual bool OnActivate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnDeactivate

Виртуальный обработчик внутреннего события "Deactivate" (деактивизация) элемента управления CBmpButton.

```
virtual bool OnDeactivate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMouseDown

Виртуальный обработчик события мыши "MouseDown" (нажатие кнопки мыши) элемента управления CBmpButton.

```
virtual bool OnMouseDown()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMouseUp

Виртуальный обработчик события мыши "MouseUp" (отпускание кнопки мыши) элемента управления CBitmap.

```
virtual bool OnMouseUp()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Класс CButton

Класс CButton является классом простого элемента управления на основе объекта "Кнопка".

### Описание

Класс CButton предназначен для создания простейших кнопок.

### Декларация

```
class CButton : public CWndObj
```

### Заголовок

```
#include <Controls\Button.mqh>
```

### Иерархия наследования

```
CObject  
  CWnd  
    CWndObj  
      CButton
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает элемент управления

Свойства	
<a href="#"><u>Pressed</u></a>	Получает/устанавливает значение свойства "Pressed"
<a href="#"><u>Locking</u></a>	Получает/устанавливает значение свойства "Locking"
<b>Обработка изменения параметров</b>	
<a href="#"><u>OnSetText</u></a>	Обработчик события "SetText" элемента управления
<a href="#"><u>OnSetColor</u></a>	Обработчик события "SetColor" элемента управления
<a href="#"><u>OnSetColorBackground</u></a>	Обработчик события "SetColorBackground" элемента управления
<a href="#"><u>OnSetColorBorder</u></a>	Обработчик события "SetColorBorder" элемента управления
<a href="#"><u>OnSetFont</u></a>	Обработчик события "SetFont" элемента управления
<a href="#"><u>OnFontSize</u></a>	Обработчик события "FontSize" элемента управления
<b>Обработка внутренних событий</b>	
<a href="#"><u>OnCreate</u></a>	Обработчик внутреннего события "Create" элемента управления
<a href="#"><u>OnShow</u></a>	Обработчик внутреннего события "Show" элемента управления
<a href="#"><u>OnHide</u></a>	Обработчик внутреннего события "Hide" элемента управления
<a href="#"><u>OnMove</u></a>	Обработчик внутреннего события "Move" элемента управления
<a href="#"><u>OnResize</u></a>	Обработчик внутреннего события "Resize" элемента управления
<a href="#"><u>OnMouseDown</u></a>	Обработчик внутреннего события "MouseDown" элемента управления
<a href="#"><u>OnOnMouseUp</u></a>	Обработчик внутреннего события "MouseUp" элемента управления

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

**Методы унаследованные от CWnd**

Destroy, OnMouseEvent, Name, ControlsTotal, Control, ControlFind, Rect, Left, Left, Top, Top, Right, Right, Bottom, Bottom, Width, Width, Height, Height, Size, Size, Size, Move, Move, Shift, Contains, Contains, Alignment, Align, Id, Id, IsEnabled, Enable, Disable, IsVisible, Visible, Show, Hide, IsActive, Activate, Deactivate, StateFlags, StateFlags, StateFlagsSet, StateFlagsReset, PropFlags, PropFlags, PropFlagsSet, PropFlagsReset, MouseX, MouseX, MouseY, MouseY, MouseY, MouseFlags, MouseFlags, MouseFocusKill, BringToTop

### Методы унаследованные от CWndObj

OnEvent, Text, Text, Color, Color, ColorBackground, ColorBackground, ColorBorder, ColorBorder, Font, Font, FontSize, FontSize, ZOrder, ZOrder

### Пример создания панели с кнопкой:

```

//+-----+
//|                                         ControlsButton.mqh |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Панель индикации и диалогов управления. Демонстрация работы класса CControlsDialog."
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
//+-----+
//| defines
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)      // indent from left (with allowance)
#define INDENT_TOP           (11)      // indent from top (with allowance)
#define INDENT_RIGHT          (11)      // indent from right (with allowance)
#define INDENT_BOTTOM          (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)     // size by X coordinate
#define BUTTON_HEIGHT          (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)     // size by X coordinate
#define LIST_HEIGHT             (179)     // size by Y coordinate
#define RADIO_HEIGHT            (56)      // size by Y coordinate
#define CHECK_HEIGHT            (93)      // size by Y coordinate
//+-----+
//| Class CControlsDialog
//| Usage: main dialog of the Controls application
//+-----+
class CControlsDialog : public CAppDialog

```

```

{
private:
    CButton           m_button1;                      // the button object
    CButton           m_button2;                      // the button object
    CButton           m_button3;                      // the fixed button object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool      Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool              CreateButton1(void);
    bool              CreateButton2(void);
    bool              CreateButton3(void);
    //--- handlers of the dependent controls events
    void              OnClickButton1(void);
    void              OnClickButton2(void);
    void              OnClickButton3(void);
};

//+-----+
//| Event Handling
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CLICK,m_button1,OnClickButton1)
ON_EVENT(ON_CLICK,m_button2,OnClickButton2)
ON_EVENT(ON_CLICK,m_button3,OnClickButton3)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor
//+-----+
CControlsDialog::CControlsDialog(void)
{
}

//+-----+
//| Destructor
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
}

```

```

if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
    return(false);
//--- create dependent controls
if(!CreateButton1())
    return(false);
if(!CreateButton2())
    return(false);
if(!CreateButton3())
    return(false);
//--- succeed
return(true);
}

//+-----+
//| Create the "Button1" button |
//+-----+
bool CControlsDialog::CreateButton1(void)
{
//--- coordinates
int x1=INDENT_LEFT;
int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
int x2=x1+BUTTON_WIDTH;
int y2=y1+BUTTON_HEIGHT;
//--- create
if(!m_button1.Create(m_chart_id,m_name+"Button1",m_subwin,x1,y1,x2,y2))
    return(false);
if(!m_button1.Text("Button1"))
    return(false);
if(!Add(m_button1))
    return(false);
//--- succeed
return(true);
}

//+-----+
//| Create the "Button2" button |
//+-----+
bool CControlsDialog::CreateButton2(void)
{
//--- coordinates
int x1=INDENT_LEFT+(BUTTON_WIDTH+CONTROLS_GAP_X);
int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
int x2=x1+BUTTON_WIDTH;
int y2=y1+BUTTON_HEIGHT;
//--- create
if(!m_button2.Create(m_chart_id,m_name+"Button2",m_subwin,x1,y1,x2,y2))
    return(false);
if(!m_button2.Text("Button2"))
    return(false);
if(!Add(m_button2))
    return(false);
}

```

```
//--- succeed
    return(true);
}
//+-----+
//| Create the "Button3" fixed button
//+-----+
bool CControlsDialog::CreateButton3(void)
{
//--- coordinates
    int x1=INDENT_LEFT+2*(BUTTON_WIDTH+CONTROLS_GAP_X);
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+BUTTON_WIDTH;
    int y2=y1+BUTTON_HEIGHT;
//--- create
    if(!m_button3.Create(m_chart_id,m_name+"Button3",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!m_button3.Text("Locked"))
        return(false);
    if(!Add(m_button3))
        return(false);
    m_button3.Locking(true);
//--- succeed
    return(true);
}
//+-----+
//| Event handler
//+-----+
void CControlsDialog::OnClickButton1(void)
{
    Comment(__FUNCTION__);
}
//+-----+
//| Event handler
//+-----+
void CControlsDialog::OnClickButton2(void)
{
    Comment(__FUNCTION__);
}
//+-----+
//| Event handler
//+-----+
void CControlsDialog::OnClickButton3(void)
{
    if(m_button3.Pressed())
        Comment(__FUNCTION__+" Состояние элемента управления: On");
    else
        Comment(__FUNCTION__+" Состояние элемента управления: Off");
}
//+-----+
```

```
//| Global Variables
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- create application dialog
if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
    return(INIT_FAILED);
//--- run application
ExtDialog.Run();
//--- succeed
return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- очистим комментарии
Comment("");
//--- destroy dialog
ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function
//+-----+
void OnChartEvent(const int id,           // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## Create

Создает элемент управления CButton.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Pressed (метод Get)

Получает значение свойства "Pressed" (состояние) элемента управления CButton.

```
bool Pressed()
```

### Возвращаемое значение

Свойство "Pressed" элемента управления CButton.

## Pressed (метод Set)

Устанавливает значение свойства "Pressed" (состояние) элемента управления CButton.

```
bool Pressed(  
    const bool pressed // состояние  
)
```

### Параметры

*pressed*

[in] Новое состояние элемента управления CButton.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Locking (метод Get)

Получает значение свойства "Locking" (фиксация) элемента управления CButton.

```
bool Locking()
```

### Возвращаемое значение

Свойство "Locking" элемента управления CButton.

## Locking (метод Set)

Устанавливает значение свойства "Locking" (фиксация) элемента управления CButton.

```
void Locking(  
    const bool flag // значение  
)
```

### Параметры

*flag*

[in] Новое значение свойства "Locking" элемента управления CButton.

### Возвращаемое значение

Нет.

## OnSetText

Виртуальный обработчик события "SetText" (изменение свойства [OBJPROP\\_TEXT](#)) элемента управления CButton.

```
virtual bool OnSetText()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetColor

Виртуальный обработчик события "SetColor" (изменение свойства [OBJPROP\\_COLOR](#)) элемента управления CButton.

```
virtual bool OnSetColor()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetColorBackground

Виртуальный обработчик события "SetColorBackground" (изменение свойства [OBJPROP\\_BGCOLOR](#)) элемента управления CButton.

```
virtual bool OnSetColorBackground()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetColorBorder

Виртуальный обработчик события "SetColorBorder" (изменение свойства [OBJPROP\\_BORDER\\_COLOR](#)) элемента управления CButton.

```
virtual bool OnSetColorBackground()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetFont

Виртуальный обработчик события "SetFont" (изменение параметра [OBJPROP\\_FONT](#)) элемента управления CButton.

```
virtual bool OnSetFont()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetFontSize

Виртуальный обработчик события "SetFontFontSize" (изменение параметра [OBJPROP\\_FONTSIZE](#)) элемента управления CButton.

```
virtual bool OnSetFontSize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnCreate

Виртуальный обработчик внутреннего события "Create" (создание) элемента управления CButton.

```
virtual bool OnCreate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnShow

Виртуальный обработчик внутреннего события "Show" (отображение) элемента управления CButton.

```
virtual bool OnShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) элемента управления CButton.

```
virtual bool OnHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMove

Виртуальный обработчик внутреннего события "Move" (перемещение) элемента управления CButton.

```
virtual bool OnMove()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnResize

Виртуальный обработчик внутреннего события "Resize" (изменение размеров) элемента управления CButton.

```
virtual bool OnResize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMouseDown

Виртуальный обработчик события мыши "MouseDown" (нажатие кнопки мыши) элемента управления CButton.

```
virtual bool OnMouseDown()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "MouseDown" происходит при нажатии левой кнопки мыши, если указатель мыши находится на элементе управления.

## OnMouseUp

Виртуальный обработчик события мыши "MouseUp" (отпускание кнопки мыши) элемента управления CButton.

```
virtual bool OnMouseUp()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "MouseUp" происходит при отпускании левой кнопки мыши, если указатель мыши находится на элементе управления.

## Класс CEdit

Класс CEdit является классом простого элемента управления на основе объекта "Поле ввода".

### Описание

Класс CEdit предназначен для создания редактируемых текстовых полей.

### Декларация

```
class CEdit : public CWndObj
```

### Заголовок

```
#include <Controls>Edit.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndObj  
CEdit
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает элемент управления

Свойства	
<a href="#"><u>ReadOnly</u></a>	Получает/устанавливает значение свойства "ReadOnly" элемента управления
<a href="#"><u>TextAlign</u></a>	Получает/устанавливает значение свойства "TextAlign" элемента управления
<b>Обработка событий объекта</b>	
<a href="#"><u>OnObjectEndEdit</u></a>	Виртуальный обработчик события <a href="#"><u>CHARTEVENT_OBJECT_ENDEDIT</u></a> объекта графика
<b>Обработка изменения параметров</b>	
<a href="#"><u>OnSetText</u></a>	Обработчик события "SetText" элемента управления
<a href="#"><u>OnSetColor</u></a>	Обработчик события "SetColor" элемента управления
<a href="#"><u>OnSetColorBackground</u></a>	Обработчик события "SetColorBackground" элемента управления
<a href="#"><u>OnSetColorBorder</u></a>	Обработчик события "SetColorBorder" элемента управления
<a href="#"><u>OnSetFont</u></a>	Обработчик события "SetFont" элемента управления
<a href="#"><u>OnFontSize</u></a>	Обработчик события "FontSize" элемента управления
<a href="#"><u>OnSetZOrder</u></a>	Обработчик события "SetZOrder" элемента управления
<b>Обработка внутренних событий</b>	
<a href="#"><u>OnCreate</u></a>	Обработчик внутреннего события "Create" элемента управления
<a href="#"><u>OnShow</u></a>	Обработчик внутреннего события "Show" элемента управления
<a href="#"><u>OnHide</u></a>	Обработчик внутреннего события "Hide" элемента управления
<a href="#"><u>OnMove</u></a>	Обработчик внутреннего события "Move" элемента управления
<a href="#"><u>OnResize</u></a>	Обработчик внутреннего события "Resize" элемента управления
<a href="#"><u>OnChange</u></a>	Обработчик внутреннего события "Change" элемента управления

[OnClick](#)

Обработчик внутреннего события "Click" элемента управления

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)**Методы унаследованные от CWnd**

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

**Методы унаследованные от CWndObj**

[Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

**Пример создания панели с редактируемым текстовым полем:**

```

//+-----+
//|                                         ControlsEdit.mqh | 
//|                                         Copyright 2017, MetaQuotes Software Corp. | 
//|                                         https://www.mql5.com | 
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Панель индикации и диалогов управления. Демонстрация работы класса ControlsEdit."
#include <Controls\Dialog.mqh>
#include <Controls>Edit.mqh>
//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)      // indent from left (with allowance)
#define INDENT_TOP           (11)      // indent from top (with allowance)
#define INDENT_RIGHT          (11)      // indent from right (with allowance)
#define INDENT_BOTTOM          (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)     // size by X coordinate
#define BUTTON_HEIGHT          (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)     // size by X coordinate

```

```

#define LIST_HEIGHT (179) // size by Y coordinate
#define RADIO_HEIGHT (56) // size by Y coordinate
#define CHECK_HEIGHT (93) // size by Y coordinate
//+-----+
//| Class CControlsDialog
//| Usage: main dialog of the Controls application
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CEdit m_edit; // CEdit объект

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool Create(const long chart,const string name,const int subwin,const
    //--- chart event handler

protected:
    //--- create dependent controls
    bool CreateEdit(void);
};

//+-----+
//| Constructor
//+-----+
CControlsDialog::CControlsDialog(void)
{
}

//+-----+
//| Destructor
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);

    //--- create dependent controls
    if(!CreateEdit())
        return(false);

    //--- succeed
    return(true);
}
//+-----+

```

```
//| Create the display field
//+-----+
bool CControlsDialog::CreateEdit(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP;
    int x2=ClientAreaWidth()-INDENT_RIGHT;
    int y2=y1+EDIT_HEIGHT;
//--- create
    if(!m_edit.Create(m_chart_id,m_name+"Edit",m_subwin,x1,y1,x2,y2))
        return(false);
//--- разрешим редактировать содержимое
    if(!m_edit.ReadOnly(false))
        return(false);
    if(!Add(m_edit))
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Global Variables
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
//--- run application
    ExtDialog.Run();
//--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- очистим комментарии
    Comment("");
//--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function
```

```
//+-----+
void OnChartEvent(const int id,           // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,   // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## Create

Создает элемент управления CEdit.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ReadOnly (метод Get)

Получает значение свойства "ReadOnly" (только чтение) элемента управления CEdit.

```
bool ReadOnly()
```

### Возвращаемое значение

Свойство "ReadOnly" элемента управления CEdit.

## ReadOnly (метод Set)

Устанавливает новое значение свойства "ReadOnly" (только чтение) элемента управления CEdit.

```
bool ReadOnly(  
    const bool flag // значение  
)
```

### Параметры

*flag*

[in] Новое значение свойства "ReadOnly" элемента управления CEdit.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## .TextAlign (метод Get)

Получает значение свойства ".TextAlign" ([тип выравнивания текста по горизонтали](#)) элемента управления CEdit.

```
ENUM_ALIGN_MODE TextAlign() const
```

### Возвращаемое значение

Значение свойства ".TextAlign" элемента управления CEdit.

## .TextAlign (метод Set)

Устанавливает значение свойства ".TextAlign" ([тип выравнивания текста по горизонтали](#)) элемента управления CEdit.

```
bool TextAlign(  
    ENUM_ALIGN_MODE align           // значение свойства  
)
```

### Параметры

*align*  
[in] Новое значение свойства ".TextAlign".

### Возвращаемое значение

true - в случае удачи, false - если не удалось изменить свойство.

## OnObjectEndEdit

Виртуальный обработчик события [CHARTEVENT\\_OBJECT\\_ENDEDIT](#) объекта графика.

```
virtual bool OnObjectEndEdit()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetText

Виртуальный обработчик события "SetText" (изменение свойства [OBJPROP\\_TEXT](#)) элемента управления CEdit.

```
virtual bool OnSetText()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetColor

Виртуальный обработчик события "SetColor" (изменение свойства [OBJPROP\\_COLOR](#)) элемента управления CEdit.

```
virtual bool OnSetColor()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetColorBackground

Виртуальный обработчик события "SetColorBackground" (изменение свойства [OBJPROP\\_BGCOLOR](#)) элемента управления CEdit.

```
virtual bool OnSetColorBackground()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetColorBorder

Виртуальный обработчик события "SetColorBorder" (изменение свойства [OBJPROP\\_BORDER\\_COLOR](#)) элемента управления CEdit.

```
virtual bool OnSetColorBorder()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetFont

Виртуальный обработчик события "SetFont" (изменение свойства [OBJPROP\\_FONT](#)) элемента управления CEdit.

```
virtual bool OnSetFont()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetFontSize

Виртуальный обработчик события "SetFontSize" (изменение свойства [OBJPROP\\_FONTSIZE](#)) элемента управления CEdit.

```
virtual bool OnSetFontSize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetZOrder

Виртуальный обработчик события "SetZOrder" (изменение свойства [OBJPROP\\_ZORDER](#)) элемента управления CEdit.

```
virtual bool OnSetZOrder()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnCreate

Виртуальный обработчик внутреннего события "Create" (создание) элемента управления CEdit.

```
virtual bool OnCreate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnShow

Виртуальный обработчик внутреннего события "Show" (отображение) элемента управления CEdit.

```
virtual bool OnShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) элемента управления CEdit.

```
virtual bool OnHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMove

Виртуальный обработчик внутреннего события "Move" (перемещение) элемента управления CEdit.

```
virtual bool OnMove()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnResize

Виртуальный обработчик внутреннего события "Resize" (изменение размеров) элемента управления CEdit.

```
virtual bool OnResize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChange

Виртуальный обработчик внутреннего события "Change" (изменение) элемента управления CEdit.

```
virtual bool OnChange()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnClick

Виртуальный обработчик внутреннего события "Click" (клик мыши) элемента управления CEdit.

```
virtual bool OnClick()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Класс CPanel

Класс CPanel является классом простого элемента управления на основе объекта "Прямоугольная метка".

### Описание

Класс CPanel предназначен для визуального объединения группы функционально связанных однородных элементов.

### Декларация

```
class CPanel : public CWndObj
```

### Заголовок

```
#include <Controls\Panel.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndObj  
CPanel
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

#### Создание

<a href="#">Create</a>	Создает элемент управления
<a href="#">Параметры объекта графика</a>	
<a href="#">BorderType</a>	Получает значение параметра "Border" (тип рамки) объекта графика
<a href="#">Обработка событий объекта</a>	
<a href="#">OnSetText</a>	Обработчик события "SetText" элемента управления
<a href="#">OnSetColorBackground</a>	Обработчик события "SetColorBackground" элемента управления
<a href="#">OnSetColorBorder</a>	Обработчик события "SetColorBorder" элемента управления
<a href="#">Обработка внутренних событий</a>	
<a href="#">OnCreate</a>	Обработчик внутреннего события "Create" элемента управления
<a href="#">OnShow</a>	Обработчик внутреннего события "Show" элемента управления
<a href="#">OnHide</a>	Обработчик внутреннего события "Hide" элемента управления
<a href="#">OnMove</a>	Обработчик внутреннего события "Move" элемента управления
<a href="#">OnResize</a>	Обработчик внутреннего события "Resize" элемента управления
<a href="#">OnChange</a>	Обработчик внутреннего события "Change" элемента управления

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CWnd

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

#### Методы унаследованные от CWndObj

[OnEvent](#), [Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

Пример создания панели с прямоугольной меткой:

```

//+-----+
//| Copyright 2017, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Панель индикации и диалогов управления. Демонстрация работы класса CControlsDialog"
#include <Controls\Dialog.mqh>
//+-----+
//| defines |
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)    // indent from left (with allowance)
#define INDENT_TOP           (11)    // indent from top (with allowance)
#define INDENT_RIGHT          (11)    // indent from right (with allowance)
#define INDENT_BOTTOM          (11)    // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)     // gap by X coordinate
#define CONTROLS_GAP_Y        (5)     // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)   // size by X coordinate
#define BUTTON_HEIGHT          (20)    // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)    // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)   // size by X coordinate
#define LIST_HEIGHT             (179)   // size by Y coordinate
#define RADIO_HEIGHT            (56)    // size by Y coordinate
#define CHECK_HEIGHT            (93)    // size by Y coordinate
//+-----+
//| Class CControlsDialog |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
public:
    CControlsDialog(void);
    ~CControlsDialog(void);

//--- create
    virtual bool Create(const long chart,const string name,const int subwin,const string title);

protected:
    //--- create dependent controls
    bool CreatePanel(void);
};

//+-----+
//| Constructor |
//+-----+

```

```
CControlsDialog::CControlsDialog(void)
{
}

//+-----+
//| Destructor
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
    //--- create dependent controls
    if(!CreatePanel())
        return(false);
    //--- succeed
    return(true);
}

//+-----+
//| Create the "CPanel"
//+-----+
bool CControlsDialog::CreatePanel(void)
{
    //--- coordinates
    int x1=20;
    int y1=20;
    int x2=ExtDialog.Width()/3;
    int y2=ExtDialog.Height()/3;
    //--- create
    if(!my_white_border.Create(0,ExtDialog.Name()+"MyWhiteBorder",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!my_white_border.ColorBackground(CONTROLS_DIALOG_COLOR_BG))
        return(false);
    if(!my_white_border.ColorBorder(CONTROLS_DIALOG_COLOR_BORDER_LIGHT))
        return(false);
    if(!ExtDialog.Add(my_white_border))
        return(false);
    my_white_border.Alignment(WND_ALIGN_CLIENT,0,0,0,0);
    //--- succeed
    return(true);
}

//+-----+
//| Global Variables
//+-----+
CControlsDialog ExtDialog;
```

```
//---  
CPanel my_white_border;           // object CPanel  
bool pause=true;                  // true - пауза  
//-----+  
//| Expert initialization function |  
//-----+  
int OnInit()  
{  
//---  
    EventSetTimer(3);  
    pause=true;  
//--- create application dialog  
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))  
        return(INIT_FAILED);  
//--- run application  
    ExtDialog.Run();  
//--- succeed  
    return(INIT_SUCCEEDED);  
}  
//-----+  
//| Expert deinitialization function |  
//-----+  
void OnDeinit(const int reason)  
{  
//--- очистим комментарии  
    Comment("");  
//--- destroy dialog  
    ExtDialog.Destroy(reason);  
}  
//-----+  
//| Expert chart event function |  
//-----+  
void OnChartEvent(const int id,          // event ID  
                  const long& lparam,   // event parameter of the long type  
                  const double& dparam, // event parameter of the double type  
                  const string& sparam) // event parameter of the string type  
{  
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);  
}  
//-----+  
//| Timer |  
//-----+  
void OnTimer()  
{  
    pause=!pause;  
}
```

## Create

Создает элемент управления CPanel.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## BorderType (метод Get)

Получает значение параметра "Border" (тип рамки) объекта графика.

```
ENUM_BORDER_TYPE BorderType()
```

### Возвращаемое значение

Значение параметра "Border".

## BorderType (метод Set)

Устанавливает значение параметра "Border" (тип рамки) объекта графика.

```
bool BorderType(  
    const ENUM_BORDER_TYPE type // значение  
)
```

### Параметры

*type*  
[in] Новое значение параметра "Border".

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnSetText

Виртуальный обработчик события "SetText" (изменение свойства [OBJPROP\\_TEXT](#)) элемента управления CPanel.

```
virtual bool OnSetText()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetColorBackground

Виртуальный обработчик события "SetColorBackground" (изменение свойства [OBJPROP\\_BGCOLOR](#)) элемента управления CPanel.

```
virtual bool OnSetColorBackground()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnSetColorBorder

Виртуальный обработчик события "SetColorBorder" (изменение свойства [OBJPROP\\_BORDER\\_COLOR](#)) элемента управления CPanel.

```
virtual bool OnSetColorBorder()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnCreate

Виртуальный обработчик внутреннего события "Create" (создание) элемента управления CPanel.

```
virtual bool OnCreate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnShow

Виртуальный обработчик внутреннего события "Show" (отображение) элемента управления CPanel.

```
virtual bool OnShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) элемента управления CPanel.

```
virtual bool OnHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMove

Виртуальный обработчик внутреннего события "Move" (перемещение) элемента управления CPanel.

```
virtual bool OnMove()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnResize

Виртуальный обработчик внутреннего события "Resize" (изменение размеров) элемента управления CPanel.

```
virtual bool OnResize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChange

Виртуальный обработчик внутреннего события "Change" (изменение) элемента управления CPanel.

```
virtual bool OnChange()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Класс CPicture

Класс CPicture является классом простого элемента управления на основе объекта "Графическая метка".

### Описание

Класс CPicture предназначен для создания простых графических изображений.

### Декларация

```
class CPicture : public CWndObj
```

### Заголовок

```
#include <Controls\Picture.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndObj  
CPicture
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

Создание	
----------	--

<a href="#">Create</a>	Создает элемент управления
<b>Свойства объекта графика</b>	
<a href="#">Border</a>	Получает/устанавливает ширину рамки объекта графика
<a href="#">BmpName</a>	Получает/устанавливает имя bmp-файла для отображения элемента управления
<b>Обработка внутренних событий</b>	
<a href="#">OnCreate</a>	Обработчик внутреннего события "Create" элемента управления
<a href="#">OnShow</a>	Обработчик внутреннего события "Show" элемента управления
<a href="#">OnHide</a>	Обработчик внутреннего события "Hide" элемента управления
<a href="#">OnMove</a>	Обработчик внутреннего события "Move" элемента управления
<a href="#">OnChange</a>	Обработчик внутреннего события "Change" элемента управления

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Save](#), [Load](#), [Type](#), [Compare](#)

#### Методы унаследованные от CWnd

[Destroy](#), [OnMouseEvent](#), [Name](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Move](#), [Move](#), [Shift](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [Id](#), [IsEnabled](#), [Enable](#), [Disable](#), [IsVisible](#), [Visible](#), [Show](#), [Hide](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

#### Методы унаследованные от CWndObj

[OnEvent](#), [Text](#), [Text](#), [Color](#), [Color](#), [ColorBackground](#), [ColorBackground](#), [ColorBorder](#), [ColorBorder](#), [Font](#), [Font](#), [FontSize](#), [FontSize](#), [ZOrder](#), [ZOrder](#)

#### Пример создания панели с графической меткой:

```
//+-----+
//|                                         ControlsPicture.mq5 |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Панель индикации и диалогов управления. Демонстрация работы клас
```

```

#include <Controls\Dialog.mqh>
#include <Controls\Picture.mqh>

//+-----+
//| defines
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)      // indent from left (with allowance)
#define INDENT_TOP           (11)      // indent from top (with allowance)
#define INDENT_RIGHT          (11)      // indent from right (with allowance)
#define INDENT_BOTTOM          (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)     // size by X coordinate
#define BUTTON_HEIGHT          (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)     // size by X coordinate
#define LIST_HEIGHT             (179)     // size by Y coordinate
#define RADIO_HEIGHT            (56)      // size by Y coordinate
#define CHECK_HEIGHT            (93)      // size by Y coordinate
//+-----+
//| Class CControlsDialog
//| Usage: main dialog of the Controls application
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CPicture      m_picture;           // CPicture object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

//--- create
    virtual bool      Create(const long chart,const string name,const int subwin,const
//--- chart event handler
    virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,const
protected:
    //--- create dependent controls
    bool              CreatePicture(void);
    //--- handlers of the dependent controls events
    void              OnClickPicture(void);
};

//+-----+
//| Event Handling
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)

```

```
ON_EVENT(ON_CLICK,m_picture,OnClickPicture)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreatePicture())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "Picture" |
//+-----+
bool CControlsDialog::CreatePicture(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+32;
    int y2=y1+32;
//--- create
    if(!m_picture.Create(m_chart_id,m_name+"Picture",m_subwin,x1,y1,x2,y2))
        return(false);
//--- установим имя bmp-файлов для отображения элемента управления CPicture
    m_picture.BmpName("\\\\Images\\\\euro.bmp");

    if(!Add(m_picture))
        return(false);
//--- succeed
    return(true);
}
//+-----+
```

```
//| Event handler
//+-----+
void CControlsDialog::OnClickPicture(void)
{
    Comment(__FUNCTION__);
}
//+-----+
//| Global Variables
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
//--- run application
    ExtDialog.Run();
//--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- очистим комментарии
    Comment("");
//--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function
//+-----+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## Create

Создает элемент управления CPicture.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,               // координата
    const int     y1,               // координата
    const int     x2,               // координата
    const int     y2                // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Border (метод Get)

Получает значение свойства "Border" (ширина рамки) элемента управления CPicture.

```
int Border() const
```

### Возвращаемое значение

Значение свойства "Border".

## Border (метод Set)

Устанавливает значение свойства "Border" (ширина рамки) элемента управления CPicture.

```
bool Border(  
    const int value // значение  
)
```

### Параметры

*value*

[in] Новое значение свойства "Border".

### Возвращаемое значение

true - в случае удачи, иначе - false.

## BmpName (метод Get)

Получает имя bmp-файла для отображения элемента управления CPicture.

```
string BmpName() const
```

### Возвращаемое значение

Имя bmp-файла для отображения элемента.

## BmpName (метод Set)

Устанавливает имя bmp-файла для отображения элемента управления CPicture.

```
bool BmpName(  
    const string name // имя файла  
)
```

### Параметры

*name*

[in] Имя bmp-файла для отображения элемента.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnCreate

Виртуальный обработчик внутреннего события "Create" (создание) элемента управления CPicture.

```
virtual bool OnCreate()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnShow

Виртуальный обработчик внутреннего события "Show" (отображение) элемента управления CPicture.

```
virtual bool OnShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) элемента управления CPicture.

```
virtual bool OnHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnMove

Виртуальный обработчик внутреннего события "Move" (перемещение) элемента управления CPicture.

```
virtual bool OnMove()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChange

Виртуальный обработчик внутреннего события "Change" (изменение) элемента управления CPicture.

```
virtual bool OnChange()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Класс CScroll

Класс CScroll является базовым классом для создания полос прокрутки.

### Описание

Класс CScroll является комбинированным элементом управления, содержащим базовые механизмы для создания полос прокрутки. Базовый класс непосредственно не используется, функциональную нагрузку несут только его наследники - классы [CScrollV](#) и [CScrollH](#).

### Декларация

```
class CScroll : public CWndContainer
```

### Заголовок

```
#include <Controls\Scrolls.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndContainer  
CScroll
```

### Прямые потомки

[CScrollH](#), [CScrollV](#)

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает элемент управления
Обработчики событий графика	
<a href="#">OnEvent</a>	Обрабатывает все события графика
Свойства	
<a href="#">MinPos</a>	Получает/устанавливает значение минимальной позиции
<a href="#">MaxPos</a>	Получает/устанавливает значение максимальной позиции
<a href="#">CurrPos</a>	Получает/устанавливает значение текущей позиции
Создание подчиненных элементов управления	
<a href="#">CreateBack</a>	Создает кнопку-подложку элемента управления

<a href="#">CreateInc</a>	Создает кнопку для увеличения позиции полосы прокрутки
<a href="#">CreateDec</a>	Создает кнопку для уменьшения позиции полосы прокрутки
<a href="#">CreateThumb</a>	Создает кнопку-бегунок текущей позиции полосы прокрутки
<b>Обработчики подчиненных элементов управления</b>	
<a href="#">OnClickInc</a>	Обработчик события увеличения позиции полосы прокрутки
<a href="#">OnClickDec</a>	Обработчик события уменьшения позиции полосы прокрутки
<b>Обработка внутренних событий</b>	
<a href="#">OnShow</a>	Обработчик внутреннего события "Create" элемента управления
<a href="#">OnHide</a>	Обработчик внутреннего события "Hide" элемента управления
<a href="#">OnChangePos</a>	Обработчик внутреннего события "ChangePosition" элемента управления
<b>События перемещения объекта</b>	
<a href="#">OnThumbDragStart</a>	Обработчик события "ThumbDragStart" элемента управления
<a href="#">OnThumbDragProcess</a>	Обработчик события "ThumbDragProcess" элемента управления
<a href="#">OnThumbDragEnd</a>	Обработчик события "ThumbDragEnd" элемента управления
<b>Расчет положения по координатам</b>	
<a href="#">CalcPos</a>	Получает позицию полосы прокрутки по координате

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)

**Методы унаследованные от CWnd**

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

**Методы унаследованные от CWndContainer**

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#), [Save](#), [Load](#)

## Create

Создает элемент управления CScroll.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string   name,           // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## MinPos (метод Get)

Получает значение параметра "MinPos" (минимальная позиция) элемента управления CScroll.

```
int MinPos() const
```

### Возвращаемое значение

Значение параметра "MinPos".

## MinPos (метод Set)

Устанавливает значение параметра "MinPos" (минимальная позиция) элемента управления CScroll.

```
void MinPos(  
    const int value // значение  
)
```

### Параметры

*value*

[in] Новое значение параметра "MinPos".

### Возвращаемое значение

Нет.

## MaxPos (метод Get)

Получает значение параметра "MaxPos" (максимальная позиция) элемента управления CScroll.

```
int MaxPos() const
```

### Возвращаемое значение

Значение параметра "MaxPos".

## MaxPos (метод Set)

Устанавливает значение параметра "MaxPos" (максимальная позиция) элемента управления CScroll.

```
void MaxPos(  
    const int value // значение  
)
```

### Параметры

*value*

[in] Новое значение параметра "MaxPos".

### Возвращаемое значение

Нет.

## CurrPos (метод Get)

Получает значение параметра "CurrPos" (текущая позиция) элемента управления CScroll.

```
int CurrPos() const
```

### Возвращаемое значение

Значение параметра "CurrPos".

## CurrPos (метод Set)

Устанавливает значение параметра "CurrPos" (текущая позиция) элемента управления CScroll.

```
void CurrPos(  
    const int value // значение  
)
```

### Параметры

*value*

[in] Новое значение параметра "CurrPos".

### Возвращаемое значение

Нет.

## CreateBack

Создает кнопку-подложку элемента управления CScroll.

```
virtual bool CreateBack()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateInc

Создает кнопку для увеличения позиции полосы прокрутки CScroll.

```
virtual bool CreateInc()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateDec

Создает кнопку для уменьшения позиции полосы прокрутки CScroll.

```
virtual bool CreateDec()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateThumb

Создает кнопку-бегунок текущей позиции полосы прокрутки CScroll.

```
virtual bool CreateThumb()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnClickInc

Виртуальный обработчик внутреннего события "ClickInc" (клик мыши на ползунок увеличения значения позиции) элемента управления CScroll.

```
virtual bool OnClickInc()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnClickDec

Виртуальный обработчик внутреннего события "ClickDec" (клик мыши на ползунок уменьшения значения позиции) элемента управления CScroll.

```
virtual bool OnClickDec()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnShow

Виртуальный обработчик внутреннего события "Show" (отображение) элемента управления CScroll.

```
virtual bool OnShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) элемента управления CScroll.

```
virtual bool OnHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChangePos

Виртуальный обработчик внутреннего события "ChangePos" (изменение позиции) элемента управления CScroll.

```
virtual bool OnChangePos()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnThumbDragStart

Виртуальный обработчик события "ThumbDragStart" (начало операции перетаскивания) элемента управления CScroll.

```
virtual bool OnThumbDragStart()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "ThumbDragStart" происходит при начале операции перетаскивания элемента управления.

## OnThumbDragProcess

Виртуальный обработчик события "ThumbDragProcess" (операция перетаскивания) элемента управления CScroll.

```
virtual bool OnThumbDragProcess(
    const int x,           // координата x
    const int y            // координата y
)
```

### Параметры

*x*

[in] Текущее значение координаты X курсора мыши.

*y*

[in] Текущее значение координаты Y курсора мыши.

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "ThumbDragProcess" происходит при перемещении элемента управления.

## OnThumbDragEnd

Виртуальный обработчик события "ThumbDragEnd" (завершение операции перетаскивания) элемента управления CScroll.

```
virtual bool OnThumbDragEnd()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "ThumbDragEnd" происходит по завершении операции перетаскивания элемента управления.

## CalcPos

Получает позицию полосы прокрутки элемента управления CScroll по координате.

```
virtual int CalcPos(  
    const int     coord      // координата  
)
```

### Параметры

*coord*

[in] Координата позиции прокрутки.

### Возвращаемое значение

Значение позиции полосы прокрутки.

## Класс CScrollV

Класс CScrollV является классом комбинированного элемента управления "Вертикальная полоса прокрутки".

### Описание

Класс CScrollV предназначен для создания вертикальных полос прокрутки.

### Декларация

```
class CScrollV : public CScroll
```

### Заголовок

```
#include <Controls\Scrolls.mqh>
```

### Иерархия наследования

```
CObject
  CWnd
    CWndContainer
      CScroll
        CScrollV
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

Создание подчиненных элементов управления	
<a href="#">CreateInc</a>	Создает кнопку для увеличения позиции полосы прокрутки
<a href="#">CreateDec</a>	Создает кнопку для уменьшения позиции полосы прокрутки
<a href="#">CreateThumb</a>	Создает кнопку-бегунок текущей позиции полосы прокрутки
<b>Обработка внутренних событий</b>	
<a href="#">OnResize</a>	Обработчик внутреннего события "Resize" элемента управления
<a href="#">OnChangePos</a>	Обработчик внутреннего события "ChangePosition" элемента управления
<b>События перемещения объекта</b>	
<a href="#">OnThumbDragStart</a>	Обработчик события "ThumbDragStart" элемента управления
<a href="#">OnThumbDragProcess</a>	Обработчик события "ThumbDragProcess" элемента управления
<a href="#">OnThumbDragEnd</a>	Обработчик события "ThumbDragEnd" элемента управления
<b>Расчет положения по координатам</b>	
<a href="#">CalcPos</a>	Получает позицию полосы прокрутки по координате

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)

#### Методы унаследованные от CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

#### Методы унаследованные от CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#), [Save](#), [Load](#)

#### Методы унаследованные от CScroll

[Create](#), [OnEvent](#), [MinPos](#), [MinPos](#), [MaxPos](#), [MaxPos](#), [CurrPos](#), [CurrPos](#)

Пример создания панели с вертикальным скроллом:

```

//+-----+
//|                                         ControlsScrollV.mq5 |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Панель индикации и диалогов управления. Демонстрация работы класса CControlsDialog"
#include <Controls\Dialog.mqh>
#include <Controls\Scrolls.mqh>
//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFTT          (11)      // indent from left (with allowance)
#define INDENT_TOP             (11)      // indent from top (with allowance)
#define INDENT_RIGHT           (11)      // indent from right (with allowance)
#define INDENT_BOTTOM          (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X         (5)       // gap by X coordinate
#define CONTROLS_GAP_Y         (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH           (100)     // size by X coordinate
#define BUTTON_HEIGHT          (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT             (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH             (150)     // size by X coordinate
#define LIST_HEIGHT              (179)     // size by Y coordinate
#define RADIO_HEIGHT             (56)      // size by Y coordinate
#define CHECK_HEIGHT             (93)      // size by Y coordinate
//+-----+
//| Class CControlsDialog                      |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CScrollV      m_scroll_v;           // CScrollV объект

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool    Create(const long chart,const string name,const int subwin,const string title);
    //--- chart event handler
    virtual bool    OnEvent(const int id,const long &lparam,const double &dparam,const string &sparam);

protected:
}

```

```
 //--- create dependent controls
 bool CreateScrollV(void);
 //--- handlers of the dependent controls events
 void OnScrollInc(void);
 void OnScrollDec(void);
};

//+-----+
//| Event Handling |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_SCROLL_INC,m_scroll_v,OnScrollInc)
ON_EVENT(ON_SCROLL_DEC,m_scroll_v,OnScrollDec)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}
//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateScrollV())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the CScrollsV object |
//+-----+
bool CControlsDialog::CreateScrollV(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP;
    int x2=x1+18;
    int y2=y1+LIST_HEIGHT;
//--- create
```

```

if(!m_scroll_v.Create(m_chart_id,m_name+"ScrollV",m_subwin,x1,y1,x2,y2))
    return(false);
//--- set up the scrollbar
m_scroll_v.MinPos(0);
//--- set up the scrollbar
m_scroll_v.MaxPos(10);
if(!Add(m_scroll_v))
    return(false);
Comment("Позиция полосы прокрутки ",m_scroll_v.CurrPos());
//--- succeed
return(true);
}

//+-----+
//| Event handler
//+-----+
void CControlsDialog::OnScrollInc(void)
{
    Comment("Позиция полосы прокрутки ",m_scroll_v.CurrPos());
}

//+-----+
//| Event handler
//+-----+
void CControlsDialog::OnScrollDec(void)
{
    Comment("Позиция полосы прокрутки ",m_scroll_v.CurrPos());
}

//+-----+
//| Global Variables
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- create application dialog
if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
    return(INIT_FAILED);
//--- run application
ExtDialog.Run();
//--- succeed
return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- очистим комментарии
}

```

```
Comment("");
//--- destroy dialog
ExtDialog.Destroy(reason);
}

//+-----+
//| Expert chart event function
//+-----+
void OnChartEvent(const int id,           // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,   // event parameter of the double type
                  const string& sparam)  // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## CreateInc

Создает кнопку для увеличения позиции полосы прокрутки CScrollV.

```
virtual bool CreateInc()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateDec

Создает кнопку для уменьшения позиции полосы прокрутки CScrollV.

```
virtual bool CreateDec()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateThumb

Создает кнопку-бегунок текущей позиции полосы прокрутки CScrollV.

```
virtual bool CreateThumb()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnResize

Виртуальный обработчик внутреннего события "Resize" (изменение размеров) элемента управления CScrollV.

```
virtual bool OnResize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChangePos

Виртуальный обработчик внутреннего события "ChangePos" (изменение позиции) элемента управления CScrollV.

```
virtual bool OnChangePos()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnThumbDragStart

Виртуальный обработчик события "ThumbDragStart" (начало операции перетаскивания) элемента управления CScrollV.

```
virtual bool OnThumbDragStart()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "ThumbDragStart" происходит при начале операции перетаскивания элемента управления.

## OnThumbDragProcess

Виртуальный обработчик события "ThumbDragProcess" (операция перетаскивания) элемента управления CScrollV.

```
virtual bool OnThumbDragProcess(
    const int x,           // координата x
    const int y            // координата y
)
```

### Параметры

*x*

[in] Текущее значение координаты X курсора мыши.

*y*

[in] Текущее значение координаты Y курсора мыши.

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "ThumbDragProcess" происходит при перемещении элемента управления.

## OnThumbDragEnd

Виртуальный обработчик события "ThumbDragEnd" (завершение операции перетаскивания) элемента управления CScrollView.

```
virtual bool OnThumbDragEnd()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "ThumbDragEnd" происходит по завершении операции перетаскивания элемента управления.

## CalcPos

Получает позицию полосы прокрутки элемента управления CScrollV по координате.

```
virtual int CalcPos(  
    const int      coord        // координата  
)
```

### Параметры

*coord*

[in] Координата позиции прокрутки.

### Возвращаемое значение

Значение позиции полосы прокрутки.

## Класс CScrollH

Класс CScrollH является классом комбинированного элемента управления "Горизонтальная полоса прокрутки".

### Описание

Класс CScrollH предназначен для создания горизонтальных полос прокрутки.

### Декларация

```
class CScrollH : public CScroll
```

### Заголовок

```
#include <Controls\Scrolls.mqh>
```

### Иерархия наследования

```
CObject
  CWnd
    CWndContainer
      CScroll
        CScrollH
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

Создание подчиненных элементов управления	
<a href="#">CreateInc</a>	Создает кнопку для увеличения позиции полосы прокрутки
<a href="#">CreateDec</a>	Создает кнопку для уменьшения позиции полосы прокрутки
<a href="#">CreateThumb</a>	Создает кнопку-бегунок текущей позиции полосы прокрутки
<b>Обработка внутренних событий</b>	
<a href="#">OnResize</a>	Обработчик внутреннего события "Resize" элемента управления
<a href="#">OnChangePos</a>	Обработчик внутреннего события "ChangePosition" элемента управления
<b>События перемещения объекта</b>	
<a href="#">OnThumbDragStart</a>	Обработчик события "ThumbDragStart" элемента управления
<a href="#">OnThumbDragProcess</a>	Обработчик события "ThumbDragProcess" элемента управления
<a href="#">OnThumbDragEnd</a>	Обработчик события "ThumbDragEnd" элемента управления
<b>Расчет положения по координатам</b>	
<a href="#">CalcPos</a>	Получает позицию полосы прокрутки по координате

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)

#### Методы унаследованные от CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

#### Методы унаследованные от CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#), [Save](#), [Load](#)

#### Методы унаследованные от CScroll

[Create](#), [OnEvent](#), [MinPos](#), [MinPos](#), [MaxPos](#), [MaxPos](#), [CurrPos](#), [CurrPos](#)

Пример создания панели с горизонтальным скроллом:

```

//+-----+
//|                                         ControlsScrollH.mq5 |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Панель индикации и диалогов управления. Демонстрация работы класса CControlsDialog"
#include <Controls\Dialog.mqh>
#include <Controls\Scrolls.mqh>
//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFTT          (11)      // indent from left (with allowance)
#define INDENT_TOP             (11)      // indent from top (with allowance)
#define INDENT_RIGHT           (11)      // indent from right (with allowance)
#define INDENT_BOTTOM          (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X         (5)       // gap by X coordinate
#define CONTROLS_GAP_Y         (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH           (100)     // size by X coordinate
#define BUTTON_HEIGHT          (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT             (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH             (150)     // size by X coordinate
#define LIST_HEIGHT              (179)     // size by Y coordinate
#define RADIO_HEIGHT             (56)      // size by Y coordinate
#define CHECK_HEIGHT             (93)      // size by Y coordinate
//+-----+
//| Class CControlsDialog                      |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CScrollH      m_scroll_v;           // CScrollH объект

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool    Create(const long chart,const string name,const int subwin,const string title);
    //--- chart event handler
    virtual bool    OnEvent(const int id,const long &lparam,const double &dparam,const string &sparam);

protected:
}

```

```

//--- create dependent controls
bool CreateScrollsH(void);
//--- handlers of the dependent controls events
void OnScrollInc(void);
void OnScrollDec(void);
};

//+-----+
//| Event Handling |
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_SCROLL_INC,m_scroll_v,OnScrollInc)
ON_EVENT(ON_SCROLL_DEC,m_scroll_v,OnScrollDec)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor |
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor |
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create |
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
//--- create dependent controls
    if(!CreateScrollsH())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the CScrollsH object |
//+-----+
bool CControlsDialog::CreateScrollsH(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP;
    int x2=x1+3*BUTTON_WIDTH;
    int y2=y1+18;
//--- create

```

```

if(!m_scroll_v.Create(m_chart_id,m_name+"ScrollsH",m_subwin,x1,y1,x2,y2))
    return(false);
//--- set up the scrollbar
m_scroll_v.MinPos(0);
//--- set up the scrollbar
m_scroll_v.MaxPos(10);
if(!Add(m_scroll_v))
    return(false);
Comment("Позиция полосы прокрутки ",m_scroll_v.CurrPos());
//--- succeed
return(true);
}

//+-----+
//| Event handler
//+-----+
void CControlsDialog::OnScrollInc(void)
{
    Comment("Позиция полосы прокрутки ",m_scroll_v.CurrPos());
}

//+-----+
//| Event handler
//+-----+
void CControlsDialog::OnScrollDec(void)
{
    Comment("Позиция полосы прокрутки ",m_scroll_v.CurrPos());
}

//+-----+
//| Global Variables
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- create application dialog
if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
    return(INIT_FAILED);
//--- run application
ExtDialog.Run();
//--- succeed
return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
//--- очистим комментарии
}

```

```
Comment("");
//--- destroy dialog
ExtDialog.Destroy(reason);
}

//+-----+
//| Expert chart event function
//+-----+
void OnChartEvent(const int id,           // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,   // event parameter of the double type
                  const string& sparam)  // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## CreateInc

Создает кнопку для увеличения позиции полосы прокрутки CScrollH.

```
virtual bool CreateInc()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateDec

Создает кнопку для уменьшения позиции полосы прокрутки CScrollH.

```
virtual bool CreateDec()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateThumb

Создает кнопку-бегунок текущей позиции полосы прокрутки CScrollH.

```
virtual bool CreateThumb()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnResize

Виртуальный обработчик внутреннего события "Resize" (изменение размеров) элемента управления CScrollH.

```
virtual bool OnResize()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChangePos

Виртуальный обработчик внутреннего события "ChangePos" (изменение позиции) элемента управления CScrollH.

```
virtual bool OnChangePos()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnThumbDragStart

Виртуальный обработчик события "ThumbDragStart" (начало операции перетаскивания) элемента управления CScrollH.

```
virtual bool OnThumbDragStart()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "ThumbDragStart" происходит при начале операции перетаскивания элемента управления.

## OnThumbDragProcess

Виртуальный обработчик события "ThumbDragProcess" (операция перетаскивания) элемента управления CScrollH.

```
virtual bool OnThumbDragProcess(
    const int x,           // координата x
    const int y            // координата y
)
```

### Параметры

*x*

[in] Текущее значение координаты X курсора мыши.

*y*

[in] Текущее значение координаты Y курсора мыши.

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "ThumbDragProcess" происходит при перемещении элемента управления.

## OnThumbDragEnd

Виртуальный обработчик события "ThumbDragEnd" (завершение операции перетаскивания) элемента управления CScrollH.

```
virtual bool OnThumbDragEnd()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "ThumbDragEnd" происходит по завершении операции перетаскивания элемента управления.

## CalcPos

Получает позицию полосы прокрутки элемента управления CScrollH по координате.

```
virtual int CalcPos(  
    const int     coord      // координата  
)
```

### Параметры

*coord*

[in] Координата позиции прокрутки.

### Возвращаемое значение

Значение позиции полосы прокрутки.

## Класс CWndClient

Класс CWndClient представляет собой комбинированный элемент управления "Клиентская область" и является базовым классом для создания областей с полосами прокрутки.

### Описание

Класс CWndClient является программной реализацией функционала для создания областей с полосами прокрутки.

### Декларация

```
class CWndClient : public CWndContainer
```

### Заголовок

```
#include <Controls\WndClient.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndContainer  
CWndClient
```

### Прямые потомки

[CCheckGroup](#), [CListView](#), [CRadioGroup](#)

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает элемент управления
Обработка событий графика	
<a href="#">OnEvent</a>	Обрабатывает все события графика
Параметры	
<a href="#">ColorBackground</a>	Устанавливает цвет фона элемента
<a href="#">ColorBorder</a>	Устанавливает цвет рамки элемента
<a href="#">BorderType</a>	Устанавливает тип рамки элемента
Настройки	
<a href="#">VScrolled</a>	Получает/устанавливает признак использования вертикальной полосы прокрутки
<a href="#">HScrolled</a>	Получает/устанавливает признак использования горизонтальной полосы

	прокрутки
<b>Подчиненные элементы управления</b>	
<a href="#"><u>CreateBack</u></a>	Создает кнопку-подложку полосы прокрутки
<a href="#"><u>CreateScrollV</u></a>	Создает вертикальную полосу прокрутки
<a href="#"><u>CreateScrollH</u></a>	Создает горизонтальную полосу прокрутки
<b>Обработка внутренних событий</b>	
<a href="#"><u>OnResize</u></a>	Виртуальный обработчик внутреннего события "Resize" элемента управления
<b>Обработка событий подчиненных элементов управления</b>	
<a href="#"><u>OnVScrollShow</u></a>	Виртуальный обработчик внутреннего события "Show" подчиненного элемента управления VScroll
<a href="#"><u>OnVScrollHide</u></a>	Виртуальный обработчик внутреннего события "Hide" подчиненного элемента управления VScroll
<a href="#"><u>OnHScrollShow</u></a>	Виртуальный обработчик внутреннего события "Show" подчиненного элемента управления HScroll
<a href="#"><u>OnHScrollHide</u></a>	Виртуальный обработчик внутреннего события "Hide" подчиненного элемента управления HScroll
<a href="#"><u>OnScrollLineDown</u></a>	Виртуальный обработчик внутреннего события "ScrollLineDown" подчиненного элемента управления VScroll
<a href="#"><u>OnScrollLineUp</u></a>	Виртуальный обработчик внутреннего события "ScrollLineUp" подчиненного элемента управления VScroll
<a href="#"><u>OnScrollLineLeft</u></a>	Виртуальный обработчик внутреннего события "ScrollLineLeft" подчиненного элемента управления HScroll
<a href="#"><u>OnScrollLineRight</u></a>	Виртуальный обработчик внутреннего события "ScrollLineRight" подчиненного элемента управления HScroll
<b>Изменение размеров</b>	
<a href="#"><u>Rebound</u></a>	Устанавливает новые параметры области элемента управления из координат класса CRect

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)

#### Методы унаследованные от CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

#### Методы унаследованные от CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Enable](#), [Disable](#), [Hide](#), [Save](#), [Load](#)

## Create

Создает элемент управления CWndClient.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## ColorBackground

Устанавливает цвет фона элемента управления.

```
bool ColorBackground(
    const color value           // цвет фона
)
```

### Параметры

*value*

[in] Цвет фона элемента управления.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ColorBorder

Устанавливает цвет рамки элемента управления.

```
bool ColorBorder(
    const color value           // цвет
)
```

### Параметры

*value*

[in] Цвет рамки элемента управления.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## BorderType

Устанавливает тип рамки элемента управления.

```
bool BorderType(
    const ENUM_BORDER_TYPE type      // значение
)
```

### Параметры

*type*

[in] Тип рамки элемента управления.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## VScrolled (метод Get)

Получает признак использования вертикальной полосы прокрутки.

```
bool VScrolled()
```

### Возвращаемое значение

true, если вертикальная полоса прокрутки используется, иначе - false.

## VScrolled (метод Set)

Устанавливает признак использования вертикальной полосы прокрутки.

```
bool VScrolled(  
    const bool flag // флаг  
)
```

### Параметры

*flag*

[in] Флаг.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## HScrolled (метод Get)

Получает признак использования горизонтальной полосы прокрутки.

```
bool HScrolled()
```

### Возвращаемое значение

true, если горизонтальная полоса прокрутки используется, иначе - false.

## HScrolled (метод Set)

Устанавливает признак использования горизонтальной полосы прокрутки.

```
bool HScrolled(  
    const bool flag // флаг  
)
```

### Параметры

*flag*

[in] Флаг.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateBack

Создает кнопку-подложку полосы прокрутки.

```
virtual bool CreateBack()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateScrollV

Создает вертикальную полосу прокрутки.

```
virtual bool CreateScrollV()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateScrollH

Создает горизонтальную полосу прокрутки.

```
virtual bool CreateScrollH()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnResize

Виртуальный обработчик внутреннего события "Resize" (изменение размеров) элемента управления CWinClient.

```
virtual bool OnResize()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnVScrollShow

Виртуальный обработчик внутреннего события "Show" (отображение) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnVScrollShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnVScrollHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnVScrollHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnHScrollShow

Виртуальный обработчик внутреннего события "Show" (отображение) подчиненного элемента управления HScroll (горизонтальная полоса прокрутки).

```
virtual bool OnHScrollShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnHScrollHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) подчиненного элемента управления HScroll (горизонтальная полоса прокрутки).

```
virtual bool OnHScrollHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnScrollLineDown

Виртуальный обработчик внутреннего события "ScrollLineDown" (прокрутка на строку вниз) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnScrollLineDown()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnScrollLineUp

Виртуальный обработчик внутреннего события "ScrollLineUp" (прокрутка на строку вверх) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnScrollLineUp()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnScrollLineLeft

Виртуальный обработчик внутреннего события "ScrollLineLeft" (прокрутка влево) подчиненного элемента управления HScroll (горизонтальная полоса прокрутки).

```
virtual bool OnScrollLineLeft()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## OnScrollLineRight

Виртуальный обработчик внутреннего события "ScrollLineRight" (прокрутка вправо) подчиненного элемента управления HScroll (горизонтальная полоса прокрутки).

```
virtual bool OnScrollLineRight()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Метод базового класса не выполняет никаких действий и всегда возвращает true.

## ReBound

Устанавливает новые параметры области элемента управления CWndClient из координат класса CRect.

```
void ReBound(  
    const & CRect rect      // класс прямоугольной области  
)
```

### Возвращаемое значение

Нет.

## Класс CListView

Класс CListView является классом комбинированного элемента управления "Просмотр списка".

### Описание

Класс CListView предназначен для отображения списка элементов данных.

### Декларация

```
class CListView : public CWndClient
```

### Заголовок

```
#include <Controls\ListView.mqh>
```

### Иерархия наследования

```
CObject
  CWnd
    CWndContainer
      CWndClient
        CListView
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

#### Создание

<a href="#"><u>Create</u></a>	Создает элемент управления
<b>Обработка событий графика</b>	
<a href="#"><u>OnEvent</u></a>	Обрабатывает все события графика
<b>Настройка</b>	
<a href="#"><u>TotalView</u></a>	Установка параметра, определяющего количество отображаемых элементов списка
<b>Наполнение</b>	
<a href="#"><u>AddItem</u></a>	Добавляет элемент (строку) в список элемента управления
<b>Данные</b>	
<a href="#"><u>Select</u></a>	Устанавливает текущий элемент списка по индексу
<a href="#"><u>SelectByText</u></a>	Устанавливает текущий элемент списка по тексту
<a href="#"><u>SelectByValue</u></a>	Устанавливает текущий элемент списка по значению
<b>Данные (только чтение)</b>	
<a href="#"><u>Value</u></a>	Получает значение текущего элемента списка
<b>Подчиненные элементы управления</b>	
<a href="#"><u>CreateRow</u></a>	Создает "строку" элемента списка
<b>Обработка внутренних событий</b>	
<a href="#"><u>OnResize</u></a>	Виртуальный обработчик внутреннего события "Resize" элемента управления
<b>Обработка событий подчиненных элементов управления</b>	
<a href="#"><u>OnVScrollShow</u></a>	Виртуальный обработчик внутреннего события "Show" подчиненного элемента управления VScroll
<a href="#"><u>OnVScrollHide</u></a>	Виртуальный обработчик внутреннего события "Hide" подчиненного элемента управления VScroll
<a href="#"><u>OnScrollLineDown</u></a>	Виртуальный обработчик внутреннего события "ScrollLineDown" подчиненного элемента управления VScroll
<a href="#"><u>OnScrollLineUp</u></a>	Виртуальный обработчик внутреннего события "ScrollLineUp" подчиненного элемента управления VScroll

<a href="#">OnItemClick</a>	Виртуальный обработчик внутреннего события "ItemClick" на заданной строке элемента управления
<a href="#">Перерисовка</a>	
<a href="#">Redraw</a>	Осуществляет перерисовку элемента управления
<a href="#">RowState</a>	Осуществляет изменение состояния заданной строки элемента управления
<a href="#">CheckView</a>	Осуществляет проверку "видимости" выбранной строки элемента управления

**Методы унаследованные от CObject**[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)**Методы унаследованные от CWnd**

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

**Методы унаследованные от CWndContainer**

[OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Enable](#), [Disable](#), [Hide](#), [Save](#), [Load](#)

**Методы унаследованные от CWndClient**

[ColorBackground](#), [ColorBorder](#), [BorderType](#), [VScrolled](#), [VScrolled](#), [HScrolled](#), [HScrolled](#), [Id](#)

**Пример создания панели со списком значений:**

```

//+-----+
//|                                         ControlsListView.mq5 |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Панель индикации и диалогов управления. Демонстрация работы класса ListView."
#include <Controls\Dialog.mqh>
#include <Controls\ListView.mqh>
//+-----+
//| defines |
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)           // indent from left (with allowance for alignment)

```

```

#define INDENT_TOP (11) // indent from top (with allowance)
#define INDENT_RIGHT (11) // indent from right (with allowance)
#define INDENT_BOTTOM (11) // indent from bottom (with allowance)
#define CONTROLS_GAP_X (5) // gap by X coordinate
#define CONTROLS_GAP_Y (5) // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH (100) // size by X coordinate
#define BUTTON_HEIGHT (20) // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT (20) // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH (150) // size by X coordinate
#define LIST_HEIGHT (179) // size by Y coordinate
#define RADIO_HEIGHT (56) // size by Y coordinate
#define CHECK_HEIGHT (93) // size by Y coordinate
//+-----+
//| Class CControlsDialog
//| Usage: main dialog of the Controls application
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CListView m_list_view; // CListView объект

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool CreateListView(void);
    //--- handlers of the dependent controls events
    void OnChangeListView(void);
};

//+-----+
//| Event Handling
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_list_view,OnChangeListView)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor
//+-----+
CControlsDialog::CControlsDialog(void)
{

```

```
    }

//+-----+
//| Destructor
//+-----+

CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create
//+-----+

bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);

    //--- create dependent controls
    if(!CreateListView())
        return(false);

    //--- succeed
    return(true);
}

//+-----+
//| Create the "ListView" element
//+-----+

bool CControlsDialog::CreateListView(void)
{
    //--- coordinates
    int x1=INDENT_LEFT+GROUP_WIDTH+2*CONTROLS_GAP_X;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+  

        (BUTTON_HEIGHT+CONTROLS_GAP_Y)+  

        (EDIT_HEIGHT+2*CONTROLS_GAP_Y);
    int x2=x1+GROUP_WIDTH;
    int y2=y1+LIST_HEIGHT-CONTROLS_GAP_Y;

    //--- create
    if(!m_list_view.Create(m_chart_id,m_name+"ListView",m_subwin,x1,y1,x2,y2))
        return(false);

    if(!Add(m_list_view))
        return(false);

    //--- fill out with strings
    for(int i=0;i<16;i++)
        if(!m_list_view.AddItem("Item "+IntegerToString(i)))
            return(false);

    //--- succeed
    return(true);
}

//+-----+
//| Event handler
//+-----+

void CControlsDialog::OnChangeListView(void)
{
```

```
Comment(__FUNCTION__+" \\""+m_list_view.Select()+"\\"");

}

//+-----+
//| Global Variables
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
    //--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    //--- run application
    ExtDialog.Run();
    //--- succeed
    return(INIT_SUCCEEDED);
}
//+-----+
//| Expert deinitialization function
//+-----+
void OnDeinit(const int reason)
{
    //--- очистим комментарии
    Comment("");
    //--- destroy dialog
    ExtDialog.Destroy(reason);
}
//+-----+
//| Expert chart event function
//+-----+
void OnChartEvent(const int id,           // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## Create

Создает элемент управления CListView.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## TotalView

Устанавливает количество видимых элементов списка элемента управления CListView.

```
bool TotalView(
    const int value           // количество видимых элементов
)
```

### Параметры

*value*

[in] Количество видимых элементов списка.

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Количество отображаемых элементов списка можно установить только один раз.

## AddItem

Добавляет элемент (строку) в список элемента управления CListView.

```
bool AddItem(  
    const string item,      // текст  
    const long   value     // значение  
)
```

### Параметры

*item*  
[in] Текст.

*value*  
[in] Значение.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Select

Устанавливает текущий элемент списка по индексу.

```
bool Select(  
    const int index // индекс  
)
```

### Параметры

*index*

[in] Индекс элемента.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## SelectByText

Устанавливает текущий элемент списка по тексту.

```
bool SelectByText(
    const string text      // текст
)
```

### Параметры

*text*

[in] Текст элемента.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## SelectByValue

Устанавливает текущий элемент списка по значению.

```
bool SelectByValue(
    const long value      // значение
)
```

### Параметры

*value*

[in] Значение.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Value

Получает значение текущего элемента списка.

```
long Value()
```

### Возвращаемое значение

Значение текущего элемента списка.

## CreateRow

Создает "строку" элемента списка.

```
bool CreateRow(
    const int index      // индекс
)
```

### Параметры

*index*

[in] Индекс элемента.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnResize

Виртуальный обработчик внутреннего события "Resize" (изменение размеров) элемента управления CListView.

```
virtual bool OnResize()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnVScrollShow

Виртуальный обработчик внутреннего события "Show" (отображение) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnVScrollShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnVScrollHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnVScrollHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnScrollLineDown

Виртуальный обработчик внутреннего события "ScrollLineDown" (прокрутка на строку вниз) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnScrollLineDown()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnScrollLineUp

Виртуальный обработчик внутреннего события "ScrollLineUp" (прокрутка на строку вверх) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnScrollLineUp()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnItemClick

Виртуальный обработчик внутреннего события "ItemClick" (клик мыши) на заданной строке элемента управления CListView.

```
virtual bool OnItemClick()  
    const int    index      // индекс  
    )
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Redraw

Осуществляет перерисовку элемента управления CListView.

```
bool Redraw()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## RowState

Осуществляет изменение состояния заданной строки элемента управления CListView.

```
bool RowState(  
    const int    index      // индекс  
    const bool   select     // состояние  
)
```

### Параметры

*index*

[in] Индекс строки.

*select*

[in] Состояние строки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CheckView

Осуществляет проверку "видимости" выбранной строки элемента управления CListView.

```
bool CheckView()
```

### Возвращаемое значение

true, если выделенная строка отображается, иначе - false.

## Класс CComboBox

Класс CComboBox является классом комбинированного элемента управления "Поле с выпадающим списком".

### Описание

Класс CComboBox представляет собой элемент управления с раскрывающимся списком, предназначенный для выбора значения.

### Декларация

```
class CComboBox : public CWndContainer
```

### Заголовок

```
#include <Controls\ComboBox.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndContainer  
CComboBox
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

#### Создание

<a href="#"><u>Create</u></a>	Создает элемент управления
<b>Обработка событий графика</b>	
<a href="#"><u>OnEvent</u></a>	Обрабатывает все события графика
<b>Наполнение</b>	
<a href="#"><u>AddItem</u></a>	Добавляет элемент (строку) в список элемента управления
<b>Настройка</b>	
<a href="#"><u>ListViewItems</u></a>	Устанавливает количество элементов выпадающего списка элемента управления
<b>Данные</b>	
<a href="#"><u>Select</u></a>	Устанавливает текущий элемент списка по индексу
<a href="#"><u>SelectByText</u></a>	Устанавливает текущий элемент списка по тексту
<a href="#"><u>SelectByValue</u></a>	Устанавливает текущий элемент списка по значению
<b>Данные (только чтение)</b>	
<a href="#"><u>Value</u></a>	Получает значение текущего элемента списка
<b>Подчиненные элементы управления</b>	
<a href="#"><u>CreateEdit</u></a>	Создает подчиненный элемент управления (строка ввода) элемента управления
<a href="#"><u>CreateButton</u></a>	Создает подчиненный элемент управления (кнопка) элемента управления
<a href="#"><u>CreateList</u></a>	Создает подчиненный элемент управления (выпадающий список) элемента управления
<b>Обработка событий подчиненных элементов управления</b>	
<a href="#"><u>OnClickEdit</u></a>	Виртуальный обработчик внутреннего события "ClickEdit" элемента управления
<a href="#"><u>OnClickButton</u></a>	Виртуальный обработчик внутреннего события "ClickButton" элемента управления
<a href="#"><u>OnChangeList</u></a>	Виртуальный обработчик внутреннего события "ChangeList" элемента управления
<b>Отображение выпадающего списка</b>	
<a href="#"><u>ListShow</u></a>	Отображает выпадающий список элемента управления

ListHide

Скрывает выпадающий список элемента управления

## Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)

## Методы унаследованные от CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

## Методы унаследованные от CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Hide](#)

## Пример создания панели с элементом управления "Поле с выпадающим списком":

```

//+-----+
//|                                                 ControlsComboBox.mqh |
//|                                                 Copyright 2015, MetaQuotes Software Corp. |
//|                                                 https://www.mql5.com |
//+-----+
#property copyright "Copyright 2015, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CComboBox"
#include <Controls\Dialog.mqh>
#include <Controls\ComboBox.mqh>
//+-----+
//| defines                                         |
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)    // indent from left (with allowance)
#define INDENT_TOP           (11)    // indent from top (with allowance)
#define INDENT_RIGHT          (11)    // indent from right (with allowance)
#define INDENT_BOTTOM          (11)    // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)     // gap by X coordinate
#define CONTROLS_GAP_Y        (5)     // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)   // size by X coordinate
#define BUTTON_HEIGHT          (20)    // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)    // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)   // size by X coordinate
#define LIST_HEIGHT             (179)   // size by Y coordinate

```

```

#define RADIO_HEIGHT (56) // size by Y coordinate
#define CHECK_HEIGHT (93) // size by Y coordinate
//+-----+
//| Class CControlsDialog
//| Usage: main dialog of the Controls application
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CComboBox m_combo_box; // CComboBox object

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool CreateComboBox(void);
    //--- handlers of the dependent controls events
    void OnChangeComboBox(void);
};

//+-----+
//| Event Handling
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_combo_box,OnChangeComboBox)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor
//+-----+
CControlsDialog::CControlsDialog(void)
{
}

//+-----+
//| Destructor
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
}

```

```

        return(false);
//--- create dependent controls
    if(!CreateComboBox())
        return(false);
//--- succeed
    return(true);
}
//+-----+
//| Create the "ComboBox" element
//+-----+
bool CControlsDialog::CreateComboBox(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+  

        (BUTTON_HEIGHT+CONTROLS_GAP_Y)+  

        (EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+GROUP_WIDTH;
    int y2=y1+EDIT_HEIGHT;
//--- create
    if(!m_combo_box.Create(m_chart_id,m_name+"ComboBox",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!Add(m_combo_box))
        return(false);
//--- fill out with strings
    for(int i=0;i<16;i++)
        if(!m_combo_box.ItemAdd("Item "+IntegerToString(i)))
            return(false);
//--- succeed
    return(true);
}
//+-----+
//| Event handler
//+-----+
void CControlsDialog::OnChangeComboBox(void)
{
    Comment(__FUNCTION__+" \\""+m_combo_box.Select()+"\\");
}
//+-----+
//| Global Variables
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))

```

```
    return(INIT_FAILED);
//--- run application
ExtDialog.Run();
//--- succeed
return(INIT_SUCCEEDED);
}

//+-----+
//| Expert deinitialization function |
//+-----+
void OnDeinit(const int reason)
{
//---

Comment("");
//--- destroy dialog
ExtDialog.Destroy(reason);
}

//+-----+
//| Expert chart event function |
//+-----+
void OnChartEvent(const int id,           // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,   // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## Create

Создает элемент управления CComboBox.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## AddItem

Добавляет элемент (строку) в список элемента управления CComboBox.

```
bool AddItem(  
    const string item,      // текст  
    const long   value     // значение  
)
```

### Параметры

*item*

[in] Текст.

*value=0*

[in] Значение.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ListViewItems

Устанавливает количество элементов выпадающего списка элемента управления CComboBox.

```
void ListViewItems(  
    const int    value      // количество элементов  
)
```

### Параметры

*value*

[in] Количество элементов выпадающего списка.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Select

Устанавливает текущий элемент списка по индексу.

```
bool Select(  
    const int index // индекс  
)
```

### Параметры

*index*

[in] Индекс элемента.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## SelectByText

Устанавливает текущий элемент списка по указанному тексту.

```
bool SelectByText(
    const string text      // текст
)
```

### Параметры

*text*

[in] Текст элемента.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## SelectByValue

Устанавливает текущий элемент списка по указанному значению.

```
bool SelectByValue(
    const long value      // значение
)
```

### Параметры

*value*

[in] Значение.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Value

Получает значение текущего элемента списка.

```
long Value()
```

### Возвращаемое значение

Значение текущего элемента списка.

## CreateEdit

Создает подчиненный элемент управления (строка ввода) элемента управления CComboBox.

```
virtual bool CreateEdit()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateButton

Создает подчиненный элемент управления (кнопка) элемента управления CComboBox.

```
virtual bool CreateButton()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateList

Создает подчиненный элемент управления (выпадающий список) элемента управления CComboBox.

```
virtual bool CreateList()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnClickEdit

Виртуальный обработчик внутреннего события "ClickEdit" (клик мыши на строке ввода) элемента управления CComboBox.

```
virtual bool OnClickEdit()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnClickButton

Виртуальный обработчик внутреннего события "ClickButton" (клик мыши на кнопке) элемента управления CComboBox.

```
virtual bool OnClickButton()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChangeList

Виртуальный обработчик внутреннего события "ChangeList" (изменение выпадающего списка) элемента управления CComboBox.

```
virtual bool OnChangeList()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## ListShow

Отображает выпадающий списокок элемента управления CComboBox.

```
virtual bool ListShow()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ListHide

Скрывает выпадающий список элемента управления CComboBox.

```
virtual bool ListHide()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Класс CCheckBox

Класс CCheckBox является классом комбинированного элемента управления "Переключатель с фиксацией".

### Описание

Класс CCheckBox предназначен для создания элемента управления, изменяющего свое состояние на противоположное при клике мыши.

### Декларация

```
class CCheckBox : public CWndContainer
```

### Заголовок

```
#include <Controls\CheckBox.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndContainer  
CCheckBox
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

#### Создание

<a href="#">Create</a>	Создает элемент управления
<a href="#">Обработка событий графика</a>	
<a href="#">OnEvent</a>	Обрабатывает все события графика
<a href="#">Настройки</a>	
<a href="#">Text</a>	Получает/устанавливает текст поясняющей надписи элемента управления
<a href="#">Color</a>	Получает/устанавливает цвет текста поясняющей надписи элемента управления
<a href="#">Состояние</a>	
<a href="#">Checked</a>	Получает/устанавливает состояние элемента управления
<a href="#">Данные</a>	
<a href="#">Value</a>	Получает/устанавливает значение, ассоциированное с элементом управления
<a href="#">Подчиненные элементы управления</a>	
<a href="#">CreateButton</a>	Создает подчиненный элемент управления (кнопка) элемента управления
<a href="#">CreateLabel</a>	Создает подчиненный элемент управления (метка поясняющей надписи) элемента управления
<a href="#">Обработка событий подчиненных элементов</a>	
<a href="#">ClickButton</a>	Виртуальный обработчик внутреннего события "ClickButton" элемента управления
<a href="#">ClickLabel</a>	Виртуальный обработчик внутреннего события "ClickLabel" элемента управления

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)

**Методы унаследованные от CWnd**

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

**Методы унаследованные от CWndContainer**

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#)

Пример создания панели с элементом управления "Переключатель с фиксацией":

```

//+-----+
//| Copyright 2017, MetaQuotes Software Corp. |
//| https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CCheckBox"
#include <Controls\Dialog.mqh>
#include <Controls\CheckBox.mqh>
//+-----+
//| defines |
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)    // indent from left (with allowance)
#define INDENT_TOP           (11)    // indent from top (with allowance)
#define INDENT_RIGHT          (11)    // indent from right (with allowance)
#define INDENT_BOTTOM          (11)    // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)     // gap by X coordinate
#define CONTROLS_GAP_Y        (5)     // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)   // size by X coordinate
#define BUTTON_HEIGHT          (20)    // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)    // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)   // size by X coordinate
#define LIST_HEIGHT             (179)   // size by Y coordinate
#define RADIO_HEIGHT            (56)    // size by Y coordinate
#define CHECK_HEIGHT             (93)    // size by Y coordinate
//+-----+
//| Class CControlsDialog |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CCheckBox      m_check_box1;           // CCheckBox object
    CCheckBox      m_check_box2;           // CCheckBox object
public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool    Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool    OnEvent(const int id,const long &lparam,const double &dparam,const

```

```

protected:
    //--- create dependent controls
    bool           CreateCheckBox1(void);
    bool           CreateCheckBox2(void);
    //--- handlers of the dependent controls events
    void          OnChangeCheckBox1(void);
    void          OnChangeCheckBox2(void);
};

//+-----+
//| Event Handling
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_check_box1,OnChangeCheckBox1)
ON_EVENT(ON_CHANGE,m_check_box2,OnChangeCheckBox2)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor
//+-----+
CControlsDialog::CControlsDialog(void)
{
}
//+-----+
//| Destructor
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
    //--- create dependent controls
    if(!CreateCheckBox1())
        return(false);
    if(!CreateCheckBox2())
        return(false);
    //--- succeed
    return(true);
}
//+-----+
//| Create the "CheckBox" element
//+-----+
bool CControlsDialog::CreateCheckBox1(void)
{
    //--- coordinates
    int x1=INDENT_LEFT;

```

```

int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+  

    (BUTTON_HEIGHT+CONTROLS_GAP_Y)+  

    (EDIT_HEIGHT+CONTROLS_GAP_Y)+  

    (EDIT_HEIGHT+CONTROLS_GAP_Y)+  

    (RADIO_HEIGHT+CONTROLS_GAP_Y);  

int x2=x1+GROUP_WIDTH;  

int y2=y1+BUTTON_HEIGHT;  

//--- create  

if(!m_check_box1.Create(m_chart_id,m_name+"CheckBox1",m_subwin,x1,y1,x2,y2))  

    return(false);  

if(!m_check_box1.Text("CheckBox1"))  

    return(false);  

if(!m_check_box1.Color(clrBlue))  

    return(false);  

if(!Add(m_check_box1))  

    return(false);  

//--- succeed  

return(true);
}  

//-----+  

//| Create the "CheckBox" element |  

//-----+  

bool CControlsDialog::CreateCheckBox2(void)  

{
//--- coordinates  

int x1=INDENT_LEFT+GROUP_WIDTH+CONTROLS_GAP_X;  

int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+  

    (BUTTON_HEIGHT+CONTROLS_GAP_Y)+  

    (EDIT_HEIGHT+CONTROLS_GAP_Y)+  

    (EDIT_HEIGHT+CONTROLS_GAP_Y)+  

    (RADIO_HEIGHT+CONTROLS_GAP_Y);  

int x2=x1+GROUP_WIDTH;  

int y2=y1+BUTTON_HEIGHT;  

//--- create  

if(!m_check_box2.Create(m_chart_id,m_name+"CheckBox2",m_subwin,x1,y1,x2,y2))  

    return(false);  

if(!m_check_box2.Text("CheckBox2"))  

    return(false);  

if(!m_check_box2.Color(clrBlue))  

    return(false);  

if(!Add(m_check_box2))  

    return(false);  

m_check_box2.Checked(true);  

Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box2.Checked()));  

//--- succeed  

return(true);
}  

//-----+  

//| Event handler |

```

```

//+
void CControlsDialog::OnChangeCheckBox1(void)
{
    Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box1.Checked()));
}

//+
//| Event handler
//+
void CControlsDialog::OnChangeCheckBox2(void)
{
    Comment(__FUNCTION__+" : Checked="+IntegerToString(m_check_box2.Checked()));
}

//+
//| Global Variables
//+
CControlsDialog ExtDialog;
//+
//| Expert initialization function
//+
int OnInit()
{
    //--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
    //--- run application
    ExtDialog.Run();
    //--- succeed
    return(INIT_SUCCEEDED);
}

//+
//| Expert deinitialization function
//+
void OnDeinit(const int reason)
{
    //---
    Comment("");
    //--- destroy dialog
    ExtDialog.Destroy(reason);
}

//+
//| Expert chart event function
//+
void OnChartEvent(const int id,          // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}

```



## Create

Создает элемент управления CCheckBox.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Text (метод Get)

Получает текст поясняющей надписи элемента управления CCheckBox.

```
string Text()
```

### Возвращаемое значение

Текст надписи.

## Text (метод Set)

Устанавливает текст поясняющей надписи элемента управления CCheckBox.

```
bool Text(  
    const string value // значение  
)
```

### Параметры

*value*

[in] Новый текст надписи.

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Текст поясняющей надписи задается путем установки значения свойства [OBJPROP\\_TEXT](#) (текст) объекта графика.

## Color (метод Get)

Получает цвет текста поясняющей надписи элемента управления CCheckBox.

```
color Color() const
```

### Возвращаемое значение

Цвет текста поясняющей надписи.

## Color (метод Set)

Устанавливает цвет поясняющей надписи элемента управления CCheckBox.

```
bool Color(  
    const color value // цвет  
)
```

### Параметры

*value*

[in] Новый цвет поясняющей надписи.

### Возвращаемое значение

true - в случае удачи, иначе - false.

### Примечание

Цвет текста поясняющей надписи задается путем установки значения свойства [OBJPROP\\_COLOR](#) (цвет) объекта графика.

## Checked (метод Get)

Получает состояние элемента управления CCheckBox.

```
bool Checked() const
```

### Возвращаемое значение

Состояние элемента управления CCheckBox.

## Checked (метод Set)

Устанавливает состояние элемента управления CCheckBox.

```
bool Checked(  
    const bool flag // состояние  
)
```

### Параметры

*flag*

[in] Новое состояние.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Value (метод Get)

Получает значение, ассоциированное с элементом управления CCheckBox.

```
int Value() const
```

### Возвращаемое значение

Значение, ассоциированное с элементом управления CCheckBox.

## Value (метод Set)

Устанавливает значение, ассоциированное с элементом управления CCheckBox.

```
void Value(  
    const int value // значение  
)
```

### Параметры

*value*

[in] Новое значение.

### Возвращаемое значение

Нет.

## CreateButton

Создает подчиненный элемент управления (кнопка) элемента управления CCheckBox.

```
virtual bool CreateButton()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateLabel

Создает подчиненный элемент управления (метка поясняющей надписи) элемента управления CComboBox.

```
virtual bool CreateLabel()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnClickButton

Виртуальный обработчик внутреннего события "ClickButton" (клик мыши на кнопке) элемента управления CCheckBox.

```
virtual bool OnClickButton()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnClickLabel

Виртуальный обработчик внутреннего события "ClickLabel" (клик мыши на надписи) элемента управления CCheckBox.

```
virtual bool OnClickLabel()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Класс CCheckGroup

Класс CCheckGroup является классом комбинированного элемента управления "Переключатель с независимой фиксацией".

### Описание

Класс CCheckGroup предназначен для создания элемента управления, позволяющего отображать и редактировать набор флагов.

### Декларация

```
class CCheckGroup : public CWndClient
```

### Заголовок

```
#include <Controls\CheckGroup.mqh>
```

### Иерархия наследования

```
CObject
  CWnd
    CWndContainer
      CWndClient
        CCheckGroup
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

<b>Создание</b>	
<a href="#">Create</a>	Создает элемент управления
<b>Обработка событий графика</b>	
<a href="#">OnEvent</a>	Обрабатывает все события графика
<b>Наполнение</b>	
<a href="#">AddItem</a>	Добавляет новый элемент в группу
<b>Данные (только чтение)</b>	
<a href="#">Value</a>	Получает значение, ассоциированное с элементом управления
<b>Подчиненные элементы управления</b>	
<a href="#">CreateButton</a>	Создает новый элемент CCheckBox в группе по указанному индексу
<b>Обработка событий подчиненных элементов</b>	
<a href="#">OnVScrollShow</a>	Виртуальный обработчик внутреннего события "Show" подчиненного элемента VScroll
<a href="#">OnVScrollHide</a>	Виртуальный обработчик внутреннего события "Hide" подчиненного элемента VScroll
<a href="#">OnScrollLineDown</a>	Виртуальный обработчик внутреннего события "ScrollLineUp" подчиненного элемента VScroll
<a href="#">OnScrollLineUp</a>	Виртуальный обработчик внутреннего события "ScrollLineDown" подчиненного элемента VScroll
<a href="#">OnChangeItem</a>	Виртуальный обработчик внутреннего события "ChangeItem" элемента управления
<b>Перерисовка</b>	
<a href="#">Redraw</a>	Перерисовывает группу элементов
<a href="#">RowState</a>	Изменяет состояние элемента группы

**Методы унаследованные от CObject**Prev, Prev, Next, Next, [Type](#), [Compare](#)**Методы унаследованные от CWnd**

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

**Методы унаследованные от CWndContainer**

[OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#),  
[Move](#), [Move](#), [Shift](#), [Enable](#), [Disable](#), [Hide](#)

**Методы унаследованные от CWndClient**

[ColorBackground](#), [ColorBorder](#), [BorderType](#), [VScrolled](#), [VScrolled](#), [HScrolled](#), [HScrolled](#), [Id](#)

Пример создания панели с элементом управления "Переключатель с независимой фиксацией":

```

//+-----+
//|                                         ControlsCheckGroup.mqh |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Control Panels and Dialogs. Demonstration class CCheckGroup"
#include <Controls\Dialog.mqh>
#include <Controls\CheckGroup.mqh>
//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)      // indent from left (with allowance)
#define INDENT_TOP           (11)      // indent from top (with allowance)
#define INDENT_RIGHT          (11)      // indent from right (with allowance)
#define INDENT_BOTTOM          (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)     // size by X coordinate
#define BUTTON_HEIGHT          (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)     // size by X coordinate
#define LIST_HEIGHT             (179)     // size by Y coordinate
#define RADIO_HEIGHT            (56)      // size by Y coordinate
#define CHECK_HEIGHT             (93)      // size by Y coordinate
//+-----+
//| Class CControlsDialog                      |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CCheckGroup      m_check_group;           // CCheckGroup object

```

```
public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool      Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,const

protected:
    //--- create dependent controls
    bool             CreateCheckGroup(void);
    //--- handlers of the dependent controls events
    void             OnChangeCheckGroup(void);
};

//+-----+
//| Event Handling
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_check_group,OnChangeCheckGroup)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor
//+-----+
CControlsDialog::CControlsDialog(void)
{
}

//+-----+
//| Destructor
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
    //--- create dependent controls
    if(!CreateCheckGroup())
        return(false);
    //--- succeed
    return(true);
}

//+-----+
//| Create the "CheckGroup" element
//+-----+
```

```

bool CControlsDialog::CreateCheckGroup (void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+  

        (BUTTON_HEIGHT+CONTROLS_GAP_Y)+  

        (EDIT_HEIGHT+CONTROLS_GAP_Y)+  

        (EDIT_HEIGHT+CONTROLS_GAP_Y)+  

        (RADIO_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+GROUP_WIDTH;
    int y2=y1+CHECK_HEIGHT;
//--- create
    if (!m_check_group.Create(m_chart_id,m_name+"CheckGroup",m_subwin,x1,y1,x2,y2))
        return (false);
    if (!Add(m_check_group))
        return (false);
//--- fill out with strings
    for(int i=0;i<5;i++)
        if (!m_check_group.AddItem("Item "+IntegerToString(i),1<<i))
            return (false);
    m_check_group.Check(0,1<<0);
    m_check_group.Check(2,1<<2);
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_check_group.Value()));
//--- succeed
    return (true);
}
//-----+
//| Event handler
//-----+
void CControlsDialog::OnChangeCheckGroup (void)
{
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_check_group.Value()));
}
//-----+
//| Global Variables
//-----+
CControlsDialog ExtDialog;
//-----+
//| Expert initialization function
//-----+
int OnInit()
{
//--- create application dialog
    if (!ExtDialog.Create(ChartID(),"Controls",0,40,40,380,344))
        return (INIT_FAILED);
//--- run application
    ExtDialog.Run();
//--- succeed
    return (INIT_SUCCEEDED);
}

```

```
    }

//+-----+
//| Expert deinitialization function           |
//+-----+

void OnDeinit(const int reason)
{
//---
    Comment("");
//--- destroy dialog
    ExtDialog.Destroy(reason);
}

//+-----+
//| Expert chart event function               |
//+-----+

void OnChartEvent(const int id,          // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## Create

Создает элемент управления CCheckGroup.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## AddItem

Добавляет в группу новый элемент (строку).

```
virtual bool AddItem(
    const string item,           // текст
    const long    value=0        // значение
)
```

### Параметры

*item*

[in] Текст поясняющей надписи добавляемого элемента управления.

*value=0*

[in] Значение, ассоциированное с добавляемым элементом управления.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Value

Получает значение, ассоциированное с элементом управления.

```
long Value()
```

### Возвращаемое значение

Значение, ассоциированное с элементом управления.

### Примечание

Значение зависит от состояний всех элементов, входящих в группу CCheckGroup.

## CreateButton

Создает новый элемент CCheckBox в группе по указанному индексу.

```
bool CreateButton(  
    int index           // индекс  
)
```

### Параметры

*index*

[in] Индекс нового элемента CheckBox в группе.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnVScrollShow

Виртуальный обработчик внутреннего события "Show" (отображение) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnVScrollShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnVScrollHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnVScrollHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnScrollLineDown

Виртуальный обработчик внутреннего события "ScrollLineDown" (прокрутка на одну позицию вниз) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnScrollLineDown()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnScrollLineUp

Виртуальный обработчик внутреннего события "ScrollLineUp" (прокрутка на одну позицию вверх) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnScrollLineUp()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChangeItem

Виртуальный обработчик внутреннего события "ChangeEvent" (изменение состояния элемента группы) элемента управления CCheckGroup.

```
virtual bool OnChangeItem(  
    const int index // индекс  
)
```

### Параметры

*index*

[in] Индекс элемента, состояние которого изменилось.

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Redraw

Перерисовывает группу элементов.

```
bool Redraw()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## RowState

Изменяет состояние элемента группы.

```
bool RowState(  
    const int    index,      // индекс  
    const bool   select     // состояние  
)
```

### Параметры

*index*

[in] Индекс элемента, состояние которого изменяется.

*select*

[in] Новое состояние элемента.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Класс CRadioButton

Класс CRadioButton является классом комбинированного элемента управления "Переключатель".

### Описание

Класс CRadioButton самостоятельного применения не имеет и используется для создания переключателей с зависимой фиксацией.

### Декларация

```
class CRadioButton : public CWndContainer
```

### Заголовок

```
#include <Controls\RadioButton.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndContainer  
CRadioButton
```

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает элемент управления
Обработка событий графика	
<a href="#">OnEvent</a>	Обрабатывает все события графика
Настройки	
<a href="#">Text</a>	Получает/устанавливает текст надписи, связанной с элементом управления
<a href="#">Color</a>	Получает/устанавливает цвет текста поясняющей надписи элемента управления
Состояние	
<a href="#">State</a>	Получает/устанавливает состояние элемента управления
Подчиненные элементы управления	
<a href="#">CreateButton</a>	Создает кнопку
<a href="#">CreateLabel</a>	Создает поясняющую надпись к кнопке
Обработка событий подчиненных элементов управления	

<a href="#">OnClickButton</a>	Виртуальный обработчик внутреннего события "ClickButton" элемента управления
<a href="#">OnClickLabel</a>	Виртуальный обработчик внутреннего события "ClickLabel" элемента управления

#### Методы унаследованные от CObject

[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)

#### Методы унаследованные от CWnd

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

#### Методы унаследованные от CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#), [Save](#), [Load](#)

## Create

Создает элемент управления CRadioButton.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Text (метод Get)

Получает текст надписи, связанной с элементом управления CRadioButton.

```
string Text() const
```

### Возвращаемое значение

Текст надписи.

## Text (метод Set)

Устанавливает текст надписи.

```
bool Text(  
    const string value // текст  
)
```

### Параметры

*value*

[in] Новый текст надписи.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Color (метод Get)

Получает цвет текста поясняющей надписи элемента управления CRadioButton.

```
color Color() const
```

### Возвращаемое значение

Цвет текста поясняющей надписи.

## Color (метод Set)

Устанавливает цвет поясняющей надписи элемента управления CRadioButton.

```
bool Color(  
    const color value // цвет  
)
```

### Параметры

*value*

[in] Новый цвет поясняющей надписи.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## State (метод Get)

Получает состояние элемента управления CRadioButton.

```
bool State() const
```

### Возвращаемое значение

Состояние кнопки.

## State (метод Set)

Устанавливает состояние элемента управления CRadioButton.

```
bool State(  
    const bool flag // состояние  
)
```

### Параметры

*flag*

[in] Новое состояние кнопки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateButton

Создает кнопку.

```
virtual bool CreateButton()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateLabel

Создает поясняющую надпись к кнопке.

```
virtual bool CreateLabel()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnClickButton

Виртуальный обработчик внутреннего события "ClickButton" (клик кнопки) элемента управления CRadioButton.

```
virtual bool OnClickButton()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnClickLabel

Виртуальный обработчик внутреннего события "ClickLabel" (клик надписи) элемента управления CRadioButton.

```
virtual bool OnClickLabel()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Класс CRadioGroup

Класс CRadioGroup является классом комбинированного элемента управления "Переключатель с зависимой фиксацией".

### Описание

Класс CRadioGroup предназначен для создания элемента управления, позволяющего отображать и редактировать поле перечислимого типа.

### Декларация

```
class CRadioGroup : public CWndClient
```

### Заголовок

```
#include <Controls\RadioGroup.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndContainer  
CWndClient  
CRadioGroup
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

<b>Создание</b>	
<a href="#">Create</a>	Создает элемент управления
<b>Обработка событий графика</b>	
<a href="#">OnEvent</a>	Обрабатывает все события графика
<b>Наполнение</b>	
<a href="#">AddItem</a>	Добавляет в группу новый элемент
<b>Данные (только чтение)</b>	
<a href="#">Value</a>	Получает значение, ассоциированное с состоянием элемента управления
<b>Подчиненные элементы управления</b>	
<a href="#">CreateButton</a>	Создает новый элемент в группе по указанному индексу
<b>Обработка событий подчиненных элементов управления</b>	
<a href="#">OnVScrollShow</a>	Виртуальный обработчик внутреннего события "Show" подчиненного элемента VScroll
<a href="#">OnVScrollHide</a>	Виртуальный обработчик внутреннего события "Hide" подчиненного элемента VScroll
<a href="#">OnScrollLineDown</a>	Виртуальный обработчик внутреннего события "ScrollLineDown" подчиненного управления VScroll
<a href="#">OnScrollLineUp</a>	Виртуальный обработчик внутреннего события "ScrollLineUp" подчиненного элемента VScroll
<a href="#">OnChangeItem</a>	Виртуальный обработчик внутреннего события "ChangeItem" элемента управления
<b>Перерисовка</b>	
<a href="#">Redraw</a>	Перерисовывает группу элементов
<a href="#">RowState</a>	Изменяет состояние элемента группы
<a href="#">Select</a>	Выбирает текущий элемент

**Методы унаследованные от CObject**Prev, Prev, Next, Next, [Type](#), [Compare](#)**Методы унаследованные от CWnd**

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlagsSet](#),

[StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

#### Методы унаследованные от CWndContainer

[OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Enable](#), [Disable](#), [Hide](#)

#### Методы унаследованные от CWndClient

[ColorBackground](#), [ColorBorder](#), [BorderType](#), [VScrolled](#), [VScrolled](#), [HScrolled](#), [HScrolled](#), [Id](#)

#### Пример создания панели с группой "радиокнопок":

```

//+-----+
//|                                         ControlsRadioGroup.mq5 |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Панель индикации и диалогов управления. Демонстрация работы класса CControlsRadioGroup."
#include <Controls\Dialog.mqh>
#include <Controls\RadioGroup.mqh>
//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFT          (11)      // indent from left (with allowance)
#define INDENT_TOP           (11)      // indent from top (with allowance)
#define INDENT_RIGHT          (11)      // indent from right (with allowance)
#define INDENT_BOTTOM          (11)      // indent from bottom (with allowance)
#define CONTROLS_GAP_X        (5)       // gap by X coordinate
#define CONTROLS_GAP_Y        (5)       // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH          (100)     // size by X coordinate
#define BUTTON_HEIGHT          (20)      // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT            (20)      // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH            (150)     // size by X coordinate
#define LIST_HEIGHT             (179)     // size by Y coordinate
#define RADIO_HEIGHT            (56)      // size by Y coordinate
#define CHECK_HEIGHT            (93)      // size by Y coordinate
//+-----+
//| Class CControlsDialog                      |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{

```

```

private:
    CRadioGroup      m_radio_group;           // CRadioGroup объект

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

    //--- create
    virtual bool     Create(const long chart,const string name,const int subwin,const
    //--- chart event handler
    virtual bool     OnEvent(const int id,const long &lparam,const double &dparam,const
    const string &sparam);

protected:
    //--- create dependent controls
    bool             CreateRadioGroup(void);
    //--- handlers of the dependent controls events
    void             OnChangeRadioGroup(void);
};

//+-----+
//| Event Handling
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
ON_EVENT(ON_CHANGE,m_radio_group,OnChangeRadioGroup)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor
//+-----+
CControlsDialog::CControlsDialog(void)
{
}

//+-----+
//| Destructor
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
    //--- create dependent controls
    if(!CreateRadioGroup())
        return(false);
    //--- succeed
    return(true);
}
//+-----+

```

```

//| Create the "RadioGroup" element
//+-----+
bool CControlsDialog::CreateRadioGroup(void)
{
//--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+  

        (BUTTON_HEIGHT+CONTROLS_GAP_Y)+  

        (EDIT_HEIGHT+CONTROLS_GAP_Y)+  

        (EDIT_HEIGHT+CONTROLS_GAP_Y);
    int x2=x1+GROUP_WIDTH;
    int y2=y1+RADIO_HEIGHT;
//--- create
    if(!m_radio_group.Create(m_chart_id,m_name+"RadioGroup",m_subwin,x1,y1,x2,y2))
        return(false);
    if(!Add(m_radio_group))
        return(false);
//--- fill out with strings
    for(int i=0;i<3;i++)
        if(!m_radio_group.AddItem("Item "+IntegerToString(i),1<<i))
            return(false);
    m_radio_group.Value(1<<2);
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_radio_group.Value()));
//--- succeed
    return(true);
}
//+-----+
//| Event handler
//+-----+
void CControlsDialog::OnChangeRadioGroup(void)
{
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_radio_group.Value()));
}
//+-----+
//| Global Variables
//+-----+
CControlsDialog ExtDialog;
//+-----+
//| Expert initialization function
//+-----+
int OnInit()
{
//--- create application dialog
    if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
        return(INIT_FAILED);
//--- run application
    ExtDialog.Run();
//--- succeed
    return(INIT_SUCCEEDED);
}

```

```
    }

//+-----+
//| Expert deinitialization function           |
//+-----+

void OnDeinit(const int reason)
{
    //--- очистим комментарии
    Comment("");
    //--- destroy dialog
    ExtDialog.Destroy(reason);
}

//+-----+
//| Expert chart event function               |
//+-----+

void OnChartEvent(const int id,          // event ID
                  const long& lparam,   // event parameter of the long type
                  const double& dparam, // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## Create

Создает элемент управления CRadioGroup.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## AddItem

Добавляет в группу новый элемент.

```
virtual bool AddItem(
    const string item,           // текст
    const long    value=0        // значение
)
```

### Параметры

*item*

[in] Текст поясняющей надписи добавляемого элемента управления.

*value=0*

[in] Значение, ассоциированное с добавляемым элементом управления.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Value

Получает значение, ассоциированное с состоянием элемента управления.

```
long Value()
```

### Возвращаемое значение

Значение, ассоциированное с состоянием элемента управления.

### Примечание

Значение зависит от состояний всех элементов CRadioButton, входящих в группу.

## CreateButton

Создает новый элемент в группе по указанному индексу.

```
bool CreateButton(
    const int index           // индекс
)
```

### Параметры

*index*

[in] Индекс нового элемента в группе.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnVScrollShow

Виртуальный обработчик внутреннего события "Show" (отображение) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnVScrollShow()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnVScrollHide

Виртуальный обработчик внутреннего события "Hide" (сокрытие) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnVScrollHide()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnScrollLineDown

Виртуальный обработчик внутреннего события "ScrollLineDown" (прокрутка на одну позицию вниз) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnScrollLineDown()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnScrollLineUp

Виртуальный обработчик внутреннего события "ScrollLineUp" (прокрутка на одну позицию вверх) подчиненного элемента управления VScroll (вертикальная полоса прокрутки).

```
virtual bool OnScrollLineUp()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## OnChangeItem

Виртуальный обработчик внутреннего события "ChangeEvent" (изменение состояния элемента группы) элемента управления CRadioGroup.

```
virtual bool OnChangeItem(  
    const int index // индекс  
)
```

### Параметры

*index*

[in] Индекс элемента, состояние которого изменилось.

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Redraw

Перерисовывает группу элементов.

```
bool Redraw()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## RowState

Изменяет состояние элемента группы.

```
bool RowState(
    const int    index,      // индекс
    const bool   select     // состояние
)
```

### Параметры

*index*

[in] Индекс элемента, состояние которого изменяется.

*select*

[in] Новое состояние элемента.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Select

Выбирает текущий элемент.

```
void Select(  
    const int index // индекс  
)
```

### Параметры

*index*

[in] Индекс выбираемого элемента.

### Возвращаемое значение

Нет.

## Класс CSpinEdit

Класс CSpinEdit является классом комбинированного элемента управления "Поле инкремента-декремента".

### Описание

Класс CSpinEdit позволяет создавать элемент управления для редактирования значения целочисленной переменной с указанным шагом в указанных пределах.

### Декларация

```
class CSpinEdit : public CWndContainer
```

### Заголовок

```
#include <Controls\SpinEdit.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndContainer  
CSpinEdit
```

Результат работы представленного ниже [кода](#):



### Методы класса по группам

<b>Создание</b>	
-----------------	--

<a href="#">Create</a>	Создает элемент управления
<b>Обработчик событий графика</b>	
<a href="#">OnEvent</a>	Обрабатывает все события графика
<b>Настройка</b>	
<a href="#">MinValue</a>	Получает/устанавливает минимальное значение элемента управления
<a href="#">MaxValue</a>	Получает/устанавливает максимальное значение элемента управления
<b>Состояние</b>	
<a href="#">Value</a>	Получает/устанавливает текущее значение элемента управления
<b>Подчиненные элементы управления</b>	
<a href="#">CreateEdit</a>	Создает подчиненный элемент управления CEdit элемента управления
<a href="#">CreateInc</a>	Создает кнопку для увеличения значения (инкремент) элемента управления
<a href="#">CreateDec</a>	Создает кнопку для уменьшения значения (декремент) элемента управления
<b>Обработка событий подчиненных элементов управления</b>	
<a href="#">OnClickInc</a>	Виртуальный обработчик внутреннего события "ClickInc" элемента управления
<a href="#">OnClickDec</a>	Виртуальный обработчик внутреннего события "ClickDec" элемента управления
<b>Обработка внутренних событий</b>	
<a href="#">OnChangeValue</a>	Виртуальный обработчик внутреннего события "ChangeValue" элемента управления

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)

**Методы унаследованные от CWnd**

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

**Методы унаследованные от CWndContainer**

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#)

Пример создания панели с прокруткой значений:

```

//+-----+
//|                                         ControlsSpinEdit.mq5 |
//|                                         Copyright 2017, MetaQuotes Software Corp. |
//|                                         https://www.mql5.com |
//+-----+
#property copyright "Copyright 2017, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property description "Панель индикации и диалогов управления. Демонстрация работы класса CControlsDialog."
#include <Controls\Dialog.mqh>
#include <Controls\SpinEdit.mqh>
//+-----+
//| defines                                     |
//+-----+
//--- indents and gaps
#define INDENT_LEFT                (11)    // indent from left (with allowance)
#define INDENT_TOP                 (11)    // indent from top (with allowance)
#define INDENT_RIGHT                (11)    // indent from right (with allowance)
#define INDENT_BOTTOM               (11)    // indent from bottom (with allowance)
#define CONTROLS_GAP_X              (5)     // gap by X coordinate
#define CONTROLS_GAP_Y              (5)     // gap by Y coordinate
//--- for buttons
#define BUTTON_WIDTH                (100)   // size by X coordinate
#define BUTTON_HEIGHT               (20)    // size by Y coordinate
//--- for the indication area
#define EDIT_HEIGHT                 (20)    // size by Y coordinate
//--- for group controls
#define GROUP_WIDTH                 (150)   // size by X coordinate
#define LIST_HEIGHT                  (179)   // size by Y coordinate
#define RADIO_HEIGHT                (56)    // size by Y coordinate
#define CHECK_HEIGHT                 (93)    // size by Y coordinate
//+-----+
//| Class CControlsDialog                      |
//| Usage: main dialog of the Controls application |
//+-----+
class CControlsDialog : public CAppDialog
{
private:
    CSpinEdit          m_spin_edit;           // CSpinEdit объект

public:
    CControlsDialog(void);
    ~CControlsDialog(void);

//--- create

```

```

virtual bool      Create(const long chart,const string name,const int subwin,const
//--- chart event handler
virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,const
const string& sparam);

protected:
    //--- create dependent controls
    bool             CreateSpinEdit(void);
    //--- handlers of the dependent controls events
    void             OnChangeSpinEdit(void);
};

//+-----+
//| Event Handling
//+-----+
EVENT_MAP_BEGIN(CControlsDialog)
    ON_EVENT(ON_CHANGE,m_spin_edit,OnChangeSpinEdit)
EVENT_MAP_END(CAppDialog)
//+-----+
//| Constructor
//+-----+
CControlsDialog::CControlsDialog(void)
{
}

//+-----+
//| Destructor
//+-----+
CControlsDialog::~CControlsDialog(void)
{
}

//+-----+
//| Create
//+-----+
bool CControlsDialog::Create(const long chart,const string name,const int subwin,const
{
    if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
        return(false);
    //--- create dependent controls
    if(!CreateSpinEdit())
        return(false);
    //--- succeed
    return(true);
}

//+-----+
//| Create the "SpinEdit" element
//+-----+
bool CControlsDialog::CreateSpinEdit(void)
{
    //--- coordinates
    int x1=INDENT_LEFT;
    int y1=INDENT_TOP+(EDIT_HEIGHT+CONTROLS_GAP_Y)+(BUTTON_HEIGHT+CONTROLS_GAP_Y);
}

```

```

int x2=x1+GROUP_WIDTH;
int y2=y1+EDIT_HEIGHT;
//--- create
if(!m_spin_edit.Create(m_chart_id,m_name+"SpinEdit",m_subwin,x1,y1,x2,y2))
    return(false);
if(!Add(m_spin_edit))
    return(false);
m_spin_edit.MinValue(10);
m_spin_edit.MaxValue(100);
m_spin_edit.Value(50);
Comment(__FUNCTION__+" : Value="+IntegerToString(m_spin_edit.Value()));
//--- succeed
return(true);
}

//-----+
//| Event handler
//-----+
void CControlsDialog::OnChangeSpinEdit(void)
{
    Comment(__FUNCTION__+" : Value="+IntegerToString(m_spin_edit.Value()));
}

//-----+
//| Global Variables
//-----+
CControlsDialog ExtDialog;
//-----+
//| Expert initialization function
//-----+
int OnInit()
{
//--- create application dialog
if(!ExtDialog.Create(0,"Controls",0,40,40,380,344))
    return(INIT_FAILED);
//--- run application
ExtDialog.Run();
//--- succeed
return(INIT_SUCCEEDED);
}

//-----+
//| Expert deinitialization function
//-----+
void OnDeinit(const int reason)
{
//--- очистим комментарии
Comment("");
//--- destroy dialog
ExtDialog.Destroy(reason);
}

```

```
//| Expert chart event function
//+-----+
void OnChartEvent(const int id,           // event ID
                  const long& lparam,    // event parameter of the long type
                  const double& dparam,  // event parameter of the double type
                  const string& sparam) // event parameter of the string type
{
    ExtDialog.ChartEvent(id,lparam,dparam,sparam);
}
```

## Create

Создает элемент управления CSpinEdit.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string   name,           // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## MinValue (метод Get)

Получает значение параметра "MinValue" (минимальное значение) элемента управления CSpinEdit.

```
int MinValue() const
```

### Возвращаемое значение

Значение параметра "MinValue".

## MinValue (метод Set)

Устанавливает значение параметра "MinValue" (минимальное значение) элемента управления CSpinEdit.

```
void MinValue(  
    const int value // значение  
)
```

### Параметры

*value*

[in] Новое значение параметра "MinValue".

### Возвращаемое значение

Нет.

## MaxValue (метод Get)

Получает значение параметра "MaxValue" (максимальное значение) элемента управления CSpinEdit.

```
int MaxValue() const
```

### Возвращаемое значение

Значение параметра "MaxValue".

## MaxValue (метод Set)

Устанавливает значение параметра "MaxValue" (максимальное значение) элемента управления CSpinEdit.

```
void MaxValue(  
    const int value // значение  
)
```

### Параметры

*value*

[in] Новое значение параметра "MaxValue".

### Возвращаемое значение

Нет.

## Value (метод Get)

Получает значение параметра "Value" (значение) элемента управления CSpinEdit.

```
int Value() const
```

### Возвращаемое значение

Значение параметра "Value".

## Value (метод Set)

Устанавливает значение параметра "Value" (значение) элемента управления CSpinEdit.

```
void Value(  
    const int value // значение  
)
```

### Параметры

*value*

[in] Новое значение параметра "Value".

### Возвращаемое значение

Нет.

## CreateEdit

Создает подчиненный элемент управления CEdit ("поле ввода") для элемента управления CSpinEdit.

```
virtual bool CreateEdit()
```

### Возвращаемое значение

true - в случае удачи, иначе - false

## CreateInc

Создает кнопку для увеличения значения (инкремент) элемента управления CSpinEdit.

```
virtual bool CreateInc()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateDec

Создает кнопку для уменьшения значения (декремент) элемента управления CSpinEdit.

```
virtual bool CreateDec()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnClickInc

Виртуальный обработчик внутреннего события "ClickInc" (клик мыши на кнопке увеличения значения) элемента управления CSpinEdit.

```
virtual bool OnClickInc()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnClickDec

Виртуальный обработчик внутреннего события "ClickDec" (клик мыши на кнопке уменьшения значения) элемента управления CSpinEdit.

```
virtual bool OnClickDec()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnChangeValue

Виртуальный обработчик внутреннего события "ChangeValue" (изменение текущего значения) элемента управления CSpinEdit.

```
virtual bool OnChangeValue()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Класс CDialog

Класс CDialog является классом комбинированного элемента управления "Диалог".

### Описание

Класс CDialog предназначен для визуального объединения группы функционально связанных разнородных элементов.

### Декларация

```
class CDialog : public CWndContainer
```

### Заголовок

```
#include <Controls\Dialog.mqh>
```

### Иерархия наследования

```
CObject  
CWnd  
CWndContainer  
CDialog
```

### Прямые потомки

[CAppDialog](#)

### Методы класса по группам

Создание	
<a href="#">Create</a>	Создает элемент управления
<a href="#">Обработка событий графика</a>	
<a href="#">OnEvent</a>	Обрабатывает все события графика
<a href="#">Настройка</a>	
<a href="#">Caption</a>	Получает/устанавливает значение свойства "Caption" (заголовок) элемента управления
<a href="#">Наполнение</a>	
<a href="#">Add</a>	Добавляет элемент управления в клиентскую область по указателю/ссылке
<a href="#">Подчиненные элементы управления</a>	
<a href="#">CreateWhiteBorder</a>	Создает подчиненный элемент (белая рамка) элемента управления
<a href="#">CreateBackground</a>	Создает подчиненный элемент (подложка) элемента управления

<a href="#">CreateCaption</a>	Создает подчиненный элемент (заголовок окна) элемента управления
<a href="#">CreateButtonClose</a>	Создает подчиненный элемент (кнопка закрытия окна) элемента управления
<a href="#">CreateClientArea</a>	Создает подчиненный элемент (клиентская область) элемента управления
Обработка событий подчиненных элементов управления	
<a href="#">OnClickCaption</a>	Виртуальный обработчик внутреннего события "ClickCaption" элемента управления
<a href="#">OnClickButtonClose</a>	Виртуальный обработчик внутреннего события "ClickButtonClose" элемента управления
Доступ к свойствам клиентской области	
<a href="#">ClientAreaVisible</a>	Устанавливает флаг видимости подчиненного элемента (клиентская область) элемента управления
<a href="#">ClientAreaLeft</a>	Получает координату X левой верхней точки клиентской области элемента управления
<a href="#">ClientAreaTop</a>	Получает координату Y левой верхней точки клиентской области элемента управления
<a href="#">ClientAreaRight</a>	Получает координату X правой нижней точки клиентской области элемента управления
<a href="#">ClientAreaBottom</a>	Получает координату Y правой нижней точки клиентской области элемента управления
<a href="#">ClientAreaWidth</a>	Получает ширину клиентской области элемента управления
<a href="#">ClientAreaHeight</a>	Получает высоту клиентской области элемента управления
Обработка перетаскивания	
<a href="#">OnDialogDragStart</a>	Виртуальный обработчик события "DialogDragStart" элемента управления
<a href="#">OnDialogDragProcess</a>	Виртуальный обработчик события "DialogDragProcess" элемента управления
<a href="#">OnDialogDragEnd</a>	Виртуальный обработчик события "DialogDragEnd" элемента управления

**Методы унаследованные от CObject**Prev, Prev, Next, Next, [Type](#), [Compare](#)**Методы унаследованные от CWnd**

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToTop](#)

#### Методы унаследованные от CWndContainer

[Destroy](#), [OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#)

## Create

Создает элемент управления CDIALOG.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&    lparam,        // параметр
    const double&  dparam,       // параметр
    const string&  sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Caption (метод Get)

Получает значение свойства "Caption" (заголовок) элемента управления CDialog.

```
string MinValue() const
```

### Возвращаемое значение

Значение свойства "Caption".

## Caption (метод Set)

Устанавливает значение свойства "Caption" (заголовок) элемента управления CDialog.

```
bool Caption(  
    const string text // текст  
)
```

### Параметры

*text*

[in] Новое значение свойства "Caption".

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Add

Добавляет элемент управления в клиентскую область по указателю.

```
bool Add(
    CWnd *control,           // указатель
)
```

### Параметры

*control*

[in] Указатель на элемент управления.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Add

Добавляет элемент управления в клиентскую область по ссылке.

```
bool Add(
    CWnd &control,           // ссылка
)
```

### Параметры

*control*

[in] Ссылка на элемент управления.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateWhiteBorder

Создает подчиненный элемент (белая рамка) элемента управления CDialog.

```
virtual bool CreateWhiteBorder()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateBackground

Создает подчиненный элемент (подложка) элемента управления CDialog.

```
virtual bool CreateBackground()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateCaption

Создает подчиненный элемент (заголовок окна) элемента управления CDialog.

```
virtual bool CreateCaption()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateButtonClose

Создает подчиненный элемент (кнопка закрытия окна) элемента управления CDialog.

```
virtual bool CreateButtonClose()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateClientArea

Создает подчиненный элемент (клиентская область) элемента управления CDialog.

```
virtual bool CreateClientArea()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnClickCaption

Виртуальный обработчик внутреннего события "ClickCaption" (клик мыши на заголовке окна) элемента управления CDialog.

```
virtual bool OnClickCaption()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## OnClickButtonClose

Виртуальный обработчик внутреннего события "ClickButtonClose" (клик мыши на кнопке закрытия окна) элемента управления CDialog.

```
virtual bool OnClickButtonClose()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ClientAreaVisible

Устанавливает флаг видимости подчиненного элемента (клиентская область) элемента управления CDialog.

```
bool ClientAreaVisible(  
    const bool visible // флаг видимости  
)
```

### Параметры

*visible*

[in] Флаг видимости.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ClientAreaLeft

Получает координату X левой верхней точки клиентской области элемента управления CDialog.

```
int ClientAreaLeft()
```

### Возвращаемое значение

Координата X левой верхней точки клиентской области.

## ClientAreaTop

Получает координату Y левой верхней точки клиентской области элемента управления CDialog.

```
int ClientAreaTop()
```

### Возвращаемое значение

Координата Y левой верхней точки клиентской области.

## ClientAreaRight

Получает координату X правой нижней точки клиентской области элемента управления CDialog.

```
int ClientAreaTop()
```

### Возвращаемое значение

Координата X правой нижней точки клиентской области.

## ClientAreaBottom

Получает координату Y правой нижней точки клиентской области элемента управления CDialog.

```
int ClientAreaBottom()
```

### Возвращаемое значение

Координата Y правой нижней точки клиентской области.

## ClientAreaWidth

Получает ширину клиентской области элемента управления CDialog.

```
int ClientAreaWidth()
```

### Возвращаемое значение

Ширина клиентской области.

## ClientAreaHeight

Получает высоту клиентской области элемента управления CDialog.

```
int ClientAreaHeight()
```

### Возвращаемое значение

Высота клиентской области.

## OnDialogDragStart

Виртуальный обработчик события "DialogDragStart" (начало операции перетаскивания) элемента управления CDialog.

```
virtual bool OnDialogDragStart()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "DialogDragStart" происходит при начале операции перетаскивания элемента управления CDialog.

## OnDialogDragProcess

Виртуальный обработчик события "DialogDragProcess" (операция перетаскивания) элемента управления CDialog.

```
virtual bool OnDialogDragProcess()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "DialogDragProcess" происходит при перемещении элемента управления CDialog.

## OnDialogDragEnd

Виртуальный обработчик события "DialogDragEnd" (завершение операции перетаскивания) элемента управления CDialog.

```
virtual bool OnDialogDragEnd()
```

### Возвращаемое значение

true - если событие обработано, иначе - false.

### Примечание

Событие "DialogDragEnd" происходит по завершении операции перетаскивания элемента управления CDialog.

## Класс CAppDialog

Класс CAppDialog является классом комбинированного элемента управления "Диалог приложения".

### Описание

Класс CAppDialog предназначен для визуального объединения группы функционально связанных разнородных элементов в пределах одной MQL5-программы.

### Декларация

```
class CAppDialog : public CDialog
```

### Заголовок

```
#include <Controls\Dialog.mqh>
```

### Иерархия наследования

```
CObject
  CWnd
    CWndContainer
      CDialog
        CAppDialog
```

### Методы класса по группам

Создание и уничтожение	
<a href="#">Create</a>	Создает элемент управления
<a href="#">Destroy</a>	Уничтожает элемент управления
Обработка событий	
<a href="#">OnEvent</a>	Обрабатывает все события
Запуск	
<a href="#">Run</a>	Запускает элемент управления
Обработка событий графика	
<a href="#">ChartEvent</a>	Обрабатывает все события графика
Настройка	
<a href="#">Minimized</a>	Устанавливает флаг минимизации элемента управления
Сохранение/восстановление состояния	
<a href="#">IniFileSave</a>	Сохраняет состояние элемента управления в файл

<a href="#">IniFileLoad</a>	Загружает состояние элемента управления из файла
<a href="#">IniFileName</a>	Устанавливает имя файла для сохранения/загрузки состояния элемента управления
<a href="#">IniFileExt</a>	Устанавливает расширение файла для сохранения/загрузки состояния элемента управления
<b>Инициализация</b>	
<a href="#">CreateCommon</a>	Метод инициализации общих настроек элемента управления
<a href="#">CreateExpert</a>	Метод инициализации настроек элемента управления для работы в эксперте
<a href="#">CreateIndicator</a>	Метод инициализации настроек элемента управления для работы в индикаторе
<b>Подчиненные элементы управления</b>	
<a href="#">CreateButtonMinMax</a>	Создает вспомогательные элементы (кнопки минимизации/восстановления окна) элемента управления
<b>Обработка событий подчиненных элементов управления</b>	
<a href="#">OnClickButtonClose</a>	Виртуальный обработчик внутреннего события "ClickButtonClose" элемента управления
<a href="#">OnClickButtonMinMax</a>	Виртуальный обработчик внутреннего события "ClickButtonMinMax" элемента управления
<b>Обработка внешних событий</b>	
<a href="#">OnAnotherApplicationClose</a>	Виртуальный обработчик внешних событий элемента управления
<b>Методы</b>	
<a href="#">Rebound</a>	Устанавливает новые параметры прямоугольной области элемента управления из координат класса CRect
<a href="#">Minimize</a>	Устанавливает окно элемента управления в минимизированное состояние
<a href="#">Maximize</a>	Устанавливает окно элемента управления в развернутое состояние
<a href="#">CreateInstanceld</a>	Создает уникальный префикс для имен объектов элемента управления

<a href="#">ProgramName</a>	Получает имя программы, в которой используется элемент управления
<a href="#">SubwinOff</a>	Получает смещение по оси Y подокна, в котором расположен элемент управления

**Методы унаследованные от CObject**

[Prev](#), [Prev](#), [Next](#), [Next](#), [Type](#), [Compare](#)

**Методы унаследованные от CWnd**

[Name](#), [ControlsTotal](#), [Control](#), [Rect](#), [Left](#), [Left](#), [Top](#), [Top](#), [Right](#), [Right](#), [Bottom](#), [Bottom](#), [Width](#), [Width](#), [Height](#), [Height](#), [Size](#), [Size](#), [Contains](#), [Contains](#), [Alignment](#), [Align](#), [Id](#), [IsEnabled](#), [IsVisible](#), [Visible](#), [IsActive](#), [Activate](#), [Deactivate](#), [StateFlags](#), [StateFlags](#), [StateFlagsSet](#), [StateFlagsReset](#), [PropFlags](#), [PropFlags](#), [PropFlagsSet](#), [PropFlagsReset](#), [MouseX](#), [MouseX](#), [MouseY](#), [MouseY](#), [MouseFlags](#), [MouseFlags](#), [MouseFocusKill](#), [BringToFront](#)

**Методы унаследованные от CWndContainer**

[OnMouseEvent](#), [ControlsTotal](#), [Control](#), [ControlFind](#), [MouseFocusKill](#), [Add](#), [Add](#), [Delete](#), [Delete](#), [Move](#), [Move](#), [Shift](#), [Id](#), [Enable](#), [Disable](#), [Show](#), [Hide](#)

**Методы унаследованные от CDialog**

[Caption](#), [Caption](#), [Add](#), [Add](#)

## Create

Создает элемент управления CAppDialog.

```
virtual bool Create(
    const long    chart,           // идентификатор графика
    const string  name,            // имя
    const int     subwin,          // подокно графика
    const int     x1,              // координата
    const int     y1,              // координата
    const int     x2,              // координата
    const int     y2               // координата
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

*x1*

[in] Значение координаты X левой верхней точки.

*y1*

[in] Значение координаты Y левой верхней точки.

*x2*

[in] Значение координаты X правой нижней точки.

*y2*

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Destroy

Метод деинициализации элемента управления CAppDialog.

```
virtual void Destroy(
    const int reason=REASON_PROGRAM // код причины
)
```

### Параметры

*reason*

[in] Код причины деинициализации. По умолчанию указывается [REASON\\_PROGRAM](#).

### Возвращаемое значение

Нет.

## OnEvent

Обрабатывает все события графика.

```
virtual bool OnEvent(
    const int      id,           // идентификатор
    const long&   lparam,        // параметр
    const double& dparam,       // параметр
    const string& sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Run

Метод запуска элемента управления CAppDialog.

```
bool Run()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## ChartEvent

Виртуальный обработчик событий элемента управления CAppDialog.

```
virtual bool ChartEvent(
    const int      id,           // идентификатор
    const long&   lparam,        // параметр
    const double& dparam,       // параметр
    const string& sparam        // параметр
)
```

### Параметры

*id*

[in] Идентификатор события.

*lparam*

[in] Ссылка на параметр события типа [long](#).

*dparam*

[in] Ссылка на параметр события типа [double](#).

*sparam*

[in] Ссылка на параметр события типа [string](#).

### Возвращаемое значение

true - если событие обработано, иначе - false.

## Minimized

Устанавливает значение флага "Minimized" (состояние окна) элемента управления CAppDialog.

```
bool Minimized(
    const bool flag      // состояние
)
```

### Параметры

*flag*

[in] Новое состояние.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## IniFileSave

Сохраняет состояние элемента управления CAppDialog в файл.

```
void IniFileSave()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## IniFileLoad

Загружает состояние элемента управления CAppDialog в файл.

```
void IniFileLoad()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## IniFileName

Получает имя файла для сохранения/загрузки состояния элемента управления CAppDialog.

```
virtual string IniFileName() const
```

### Возвращаемое значение

Имя файла для сохранения/загрузки состояния элемента управления CAppDialog.

### Примечание

Имя файла включает в себя имя индикатора/эксперта а также символ, на котором работает программа.

## IniFileExt

Получает расширение файла для сохранения/загрузки состояния элемента управления CAppDialog.

```
virtual string IniFileExt() const
```

### Возвращаемое значение

Расширение файла для сохранения/загрузки состояния элемента управления CAppDialog.

## CreateCommon

Метод инициализации общих настроек элемента управления CAppDialog.

```
bool CreateCommon(
    const long    chart,      // идентификатор графика
    const string  name,       // имя
    const int     subwin,     // подокно графика
)
```

### Параметры

*chart*

[in] Идентификатор графика, на котором создается элемент управления.

*name*

[in] Уникальное имя элемента управления.

*subwin*

[in] Подокно графика, в котором создается элемент управления.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateExpert

Метод инициализации настроек элемента управления CAppDialog для работы в эксперте.

```
bool CreateExpert(
    const int    x1,           // координата
    const int    y1,           // координата
    const int    x2,           // координата
    const int    y2            // координата
)
```

### Параметры

x1

[in] Значение координаты X левой верхней точки.

y1

[in] Значение координаты Y левой верхней точки.

x2

[in] Значение координаты X правой нижней точки.

y2

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateIndicator

Метод инициализации настроек элемента управления CAppDialog для работы в индикаторе.

```
bool CreateIndicator(
    const int    x1,           // координата
    const int    y1,           // координата
    const int    x2,           // координата
    const int    y2            // координата
)
```

### Параметры

x1

[in] Значение координаты X левой верхней точки.

y1

[in] Значение координаты Y левой верхней точки.

x2

[in] Значение координаты X правой нижней точки.

y2

[in] Значение координаты Y правой нижней точки.

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateButtonMinMax

Создает вспомогательные элементы (кнопки минимизации/восстановления) элемента управления CAppDialog.

```
virtual void CreateButtonMinMax()
```

### Возвращаемое значение

Нет.

## OnClickButtonClose

Виртуальный обработчик внутреннего события "ClickButtonClose" (клик мыши на кнопке закрытия) элемента управления CAppDialog.

```
virtual void OnClickButtonClose()
```

### Возвращаемое значение

Нет.

## OnClickButtonMinMax

Виртуальный обработчик внутреннего события "ClickButtonMinMax" (клик мыши на кнопке минимизации/восстановления) элемента управления CAppDialog.

```
virtual void OnClickButtonClose()
```

### Возвращаемое значение

Нет.

## OnAnotherApplicationClose

Виртуальный обработчик внешних событий элемента управления CAppDialog.

```
virtual void OnAnotherApplicationClose()
```

### Возвращаемое значение

Нет.

## Rebound

Устанавливает новые параметры прямоугольной области элемента управления CAppDialog из координат класса CRect.

```
bool Rebound(  
    const & CRect rect          // класс прямоугольной области  
)
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Minimize

Устанавливает окно элемента управления CAppDialog в минимизированное (свернутое) состояние.

```
virtual void Minimize()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## Maximize

Устанавливает окно элемента управления CAppDialog в развернутое состояние.

```
virtual void Maximize()
```

### Возвращаемое значение

true - в случае удачи, иначе - false.

## CreateInstanceId

Создает уникальный префикс для имен объектов элемента управления CAppDialog.

```
string CreateInstanceId()
```

### Возвращаемое значение

Префикс для имен объектов.

## ProgramName

Получает имя программы, в которой используется элемент управления CAppDialog.

```
string ProgramName()
```

### Возвращаемое значение

Имя MQL5-программы.

## SubwinOff

Получает смещение по оси Y подокна, в котором расположен элемент управления CAppDialog.

```
void SubwinOff()
```

### Возвращаемое значение

Нет.

## Переход с MQL4

Язык MQL5 является развитием своего предшественника - языка MQL4, на котором написано огромное множество индикаторов, скриптов и экспертов. Несмотря на то, что новый язык программирования максимально совместим с языком предыдущего поколения, все же есть ряд отличий между этими языками. И при переносе программ эти различия нужно знать.

В этом разделе собрана информация, которая призвана облегчить адаптацию своих кодов тем программистам, которые хорошо знают MQL4, к новому языку MQL5.

Прежде всего необходимо отметить:

- Функции `start()`, `init()` и `deinit()` отсутствуют;
- Количество индикаторных буферов не ограничено;
- Загрузка dll происходит сразу после загрузки эксперта (или любой другой mq5-программы);
- Укороченная проверка логических условий;
- При выходе за пределы массива текущее выполнение прекращается (критически - с выводом ошибки);
- Приоритет операций как в C++;
- Работает неявное приведение типов (даже из строки в число);
- Локальные переменные автоматически не инициализируются (кроме строк);
- Обычные локальные массивы уничтожаются автоматически.

### Специальные функции `init`, `start` и `deinit`

В языке MQL4 было только три предопределенные функции, которые могли присутствовать в коде индикатора, советника или скрипта (не принимая во внимание включаемые файлы \*.mqh и файлы библиотек). Этих функций нет в MQL5, но есть их аналоги. В таблице представлено примерное соответствие функций.

MQL4	MQL5
<code>init</code>	<code>OnInit</code>
<code>start</code>	<code>OnStart</code>
<code>deinit</code>	<code>OnDeinit</code>

Функции [OnInit](#) и [OnDeinit](#) выполняют ту же роль, что и функции `init` и `deinit` в MQL4 - они предназначены для размещения кода, который должен выполняться при инициализации и деинициализации mq5-программ. Вы можете либо просто переименовать эти функции соответствующим образом, либо оставить их как есть, но добавить вызов этих функций в соответствующих местах.

Пример:

```
void OnInit()
{
//--- вызовем функцию при инициализации
init();
```

```

    }

void OnDeinit(const int reason)
{
//--- вызовем функцию при deinициализации
deinit();
//---
}

```

Функция start заменяется на [OnStart](#) только в скриптах, а для эксперта и индикатора необходимо ее переименовать соответственно в [OnTick](#) и [OnCalculate](#). Именно в этих трех функциях необходимо размещать код, который должен выполняться во время работы mq5-программы:

mq5-программа	основная функция
<a href="#">скрипт</a>	OnStart
<a href="#">индикатор</a>	OnCalculate
<a href="#">эксперт</a>	OnTick

Если основная функция отсутствует в коде индикатора или скрипта, или название этой функции отличается от требуемого, то вызов такой функции не производится. То есть, если в исходном коде скрипта не будет содержаться функции OnStart, то такой код будет скомпилирован как советник.

Если в коде индикатора отсутствует функция OnCalculate, то компиляция такого индикатора невозможна.

## Предопределенные переменные

В MQL5 нет таких предопределенных переменных как Ask, Bid, Bars. Переменные Digits и Point немного изменились в написании, как показано в таблице.

MQL4	MQL5
Digits	_Digits
Point	_Point
	_LastError
	_Period
	_Symbol
	_StopFlag
	_UninitReason

## Доступ к таймсерием

В MQL5 нет предопределенных таймсерий `Open[]`, `High[]`, `Low[]`, `Close[]`, `Volume[]` и `Time[]`. Необходимую глубину таймсерии теперь можно задавать самостоятельно с помощью соответствующих [функций для доступа к таймсериям](#).

## Советники (эксперты)

Эксперты в MQL5 не обязательно должны иметь функцию-обработчик [события](#) поступления нового тика `OnTick`, как это было в MQL4 (функция `start` в MQL4 вызывается на выполнение при поступлении нового тика), потому что теперь эксперты в MQL5 могут содержать предопределенные функции-обработчики нескольких типов событий:

- [OnTick](#) - поступление нового тика;
- [OnTimer](#) - событие таймера;
- [OnTrade](#) - торговое событие;
- [OnChartEvent](#) - события ввода от клавиатуры и мышки, события перемещения графического объекта, событие окончания редактирования текста в поле ввода объекта `LabelEdit`;
- [OnBookEvent](#) - событие изменения состояния стакана цен (Depth of Market).

## Пользовательские индикаторы

В MQL4 количество индикаторных буферов ограничено и не может быть более 8. В MQL5 такого ограничения нет, но следует помнить, что каждый индикаторный буфер требует выделения определенного объема оперативной памяти под его размещение в терминале, поэтому злоупотреблять открывшейся новой возможностью все же не следует.

Кроме того, в MQL4 было всего 6 типов отрисовки пользовательского индикатора, в MQL5 теперь 18 [стилей рисования](#). И хотя названия видов отрисовки не изменились, сама идеология графического отображения индикаторов изменилась существенно.

Отличается также и направление индексации в индикаторных буферах. В MQL5 все буфера по умолчанию ведут себя как обычные массивы, то есть элемент с индексом 0 является самым старым в истории, при увеличении индекса мы продвигаемся от самых старых данных к самым последним.

Единственная функция по работе с [пользовательскими индикаторами](#), которая сохранилась из MQL4 - это [SetIndexBuffer](#). Но и ее вызов изменился, теперь требуется указать [тип данных, которые будут храниться в массиве](#), связываемом с индикаторным буфером.

Кроме того, изменились и расширились свойства пользовательских индикаторов, появились функции [доступа к таймсериям](#), поэтому необходимо полностью переосмыслить весь алгоритм расчета.

## Графические объекты

Количество графических объектов в MQL5 существенно расширилось. Кроме того, позиционирование графических объектов во времени стало возможным с точностью до секунды на графике любого периода - теперь не производится округление точек привязок графических объектов с точностью до времени открытия бара на текущем ценовом графике.

Для объектов Arrow, Text и Label появилась возможность указывать [способ привязки](#), а для объектов Label, Button, Chart, Bitmap Label и Edit есть возможность задавать [угол графика, к которому привязан объект](#).

## Список функций языка MQL5

Все функции MQL5 в алфавитном порядке.

Функция	Действие	Раздел
<a href="#">AccountInfoDouble</a>	Возвращает значение типа double соответствующего свойства счета	<a href="#">Информация о счете</a>
<a href="#">AccountInfoInteger</a>	Возвращает значение целочисленного типа (bool,int или long) соответствующего свойства счета	<a href="#">Информация о счете</a>
<a href="#">AccountInfoString</a>	Возвращает значение типа string соответствующего свойства счета	<a href="#">Информация о счете</a>
<a href="#">acos</a>	Возвращает значение арккосинуса x в радианах	<a href="#">Математические функции</a>
<a href="#">Alert</a>	Выводит сообщение в отдельном окне	<a href="#">Общие функции</a>
<a href="#">ArrayBsearch</a>	Возвращает индекс первого найденного элемента в первом измерении массива	<a href="#">Операции с массивами</a>
<a href="#">ArrayCompare</a>	Возвращает результат сравнения двух массивов <a href="#">простых типов</a> или пользовательских структур, не имеющих <a href="#">сложных объектов</a>	<a href="#">Операции с массивами</a>
<a href="#">ArrayCopy</a>	Копирует один массив в другой	<a href="#">Операции с массивами</a>
<a href="#">ArrayFill</a>	Заполняет числовой массив указанным значением	<a href="#">Операции с массивами</a>
<a href="#">ArrayFree</a>	Освобождает буфер любого динамического массива и устанавливает размер нулевого измерения в 0 (ноль)	<a href="#">Операции с массивами</a>
<a href="#">ArrayGetAsSeries</a>	Проверяет направление индексации массива	<a href="#">Операции с массивами</a>
<a href="#">ArrayInitialize</a>	Устанавливает все элементы числового массива в одну величину	<a href="#">Операции с массивами</a>

<a href="#">ArrayIsDynamic</a>	Проверяет, является ли массив динамическим	<a href="#">Операции с массивами</a>
<a href="#">ArrayIsSeries</a>	Проверяет, является ли массив таймсерией	<a href="#">Операции с массивами</a>
<a href="#">ArrayMaximum</a>	Ищет максимальный элемент в первом измерении многомерного числового массива	<a href="#">Операции с массивами</a>
<a href="#">ArrayMinimum</a>	Ищет минимальный элемент в первом измерении многомерного числового массива	<a href="#">Операции с массивами</a>
<a href="#">ArrayRange</a>	Возвращает число элементов в указанном измерении массива	<a href="#">Операции с массивами</a>
<a href="#">ArrayResize</a>	Устанавливает новый размер в первом измерении массива	<a href="#">Операции с массивами</a>
<a href="#">ArraySetAsSeries</a>	Устанавливает направление индексирования в массиве	<a href="#">Операции с массивами</a>
<a href="#">ArraySize</a>	Возвращает количество элементов в массиве	<a href="#">Операции с массивами</a>
<a href="#">ArraySort</a>	Сортирует многомерный числовой массив по возрастанию значений в первом измерении	<a href="#">Операции с массивами</a>
<a href="#">asin</a>	Возвращает значение арксинуса $x$ в радианах	<a href="#">Математические функции</a>
<a href="#">atan</a>	Возвращает арктангенс $x$ в радианах	<a href="#">Математические функции</a>
<a href="#">Bars</a>	Возвращает количество баров в истории по соответствующим символу и периоду	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">BarsCalculated</a>	Возвращает количество рассчитанных данных в индикаторном буфере или -1 в случае ошибки (данные еще не рассчитаны)	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">CalendarCountryById</a>	Получает описание страны по её идентификатору	<a href="#">Экономический календарь</a>
<a href="#">CalendarEventById</a>	Получает описание события по его идентификатору	<a href="#">Экономический календарь</a>

<a href="#">CalendarValueById</a>	Получает описание значения события по его идентификатору	<a href="#">Экономический календарь</a>
<a href="#">CalendarCountries</a>	Получает массив описаний стран, доступных в Календаре	<a href="#">Экономический календарь</a>
<a href="#">CalendarEventByCountry</a>	Получает массив описаний всех событий, доступных в Календаре, по указанному коду страны	<a href="#">Экономический календарь</a>
<a href="#">CalendarEventByCurrency</a>	Получает массив описаний всех событий, доступных в Календаре, по указанной валюте	<a href="#">Экономический календарь</a>
<a href="#">CalendarValueHistoryByEvent</a>	Получает массив значений по всем событиям на заданном диапазоне времени по идентификатору события	<a href="#">Экономический календарь</a>
<a href="#">CalendarValueHistory</a>	Получает массив значений по всем событиям на заданном диапазоне времени с фильтром по стране и/или валюте	<a href="#">Экономический календарь</a>
<a href="#">CalendarValueLastByEvent</a>	Получает массив значений события по его ID с момента состояния базы Календаря с заданным change_id	<a href="#">Экономический календарь</a>
<a href="#">CalendarValueLast</a>	Получает массив значений по всем событиям с фильтрацией по стране и/или валюте с момента состояния базы Календаря с заданным change_id	<a href="#">Экономический календарь</a>
<a href="#">ceil</a>	Возвращает ближайшее сверху целое числовое значение	<a href="#">Математические функции</a>
<a href="#">CharArrayToString</a>	Преобразует код символа (ansi) в односимвольную строку	<a href="#">Преобразование данных</a>
<a href="#">ChartApplyTemplate</a>	Применяет к указанному графику шаблон из указанного файла	<a href="#">Операции с графиками</a>
<a href="#">ChartClose</a>	Закрывает указанный график	<a href="#">Операции с графиками</a>

<a href="#">ChartFirst</a>	Возвращает идентификатор первого графика клиентского терминала	<a href="#">Операции с графиками</a>
<a href="#">ChartGetDouble</a>	Возвращает значение соответствующего свойства указанного графика	<a href="#">Операции с графиками</a>
<a href="#">ChartGetInteger</a>	Возвращает целочисленное значение соответствующего свойства указанного графика	<a href="#">Операции с графиками</a>
<a href="#">ChartGetString</a>	Возвращает строковое значение соответствующего свойства указанного графика	<a href="#">Операции с графиками</a>
<a href="#">ChartID</a>	Возвращает идентификатор текущего графика	<a href="#">Операции с графиками</a>
<a href="#">ChartIndicatorAdd</a>	Добавляет на указанное окно графика индикатор с указанным хэндлом.	<a href="#">Операции с графиками</a>
<a href="#">ChartIndicatorDelete</a>	Удаляет с указанного окна графика индикатор с указанным именем	<a href="#">Операции с графиками</a>
<a href="#">ChartIndicatorGet</a>	Возвращает хэндл индикатора с указанным коротким именем на указанном окне графика	<a href="#">Операции с графиками</a>
<a href="#">ChartIndicatorName</a>	Возвращает короткое имя индикатора по номеру в списке индикаторов на указанном окне графика.	<a href="#">Операции с графиками</a>
<a href="#">ChartIndicatorsTotal</a>	Возвращает количество всех индикаторов, присоединенных к указанному окну графика.	<a href="#">Операции с графиками</a>
<a href="#">ChartNavigate</a>	Осуществляет сдвиг указанного графика на указанное количество баров относительно указанной позиции графика	<a href="#">Операции с графиками</a>
<a href="#">ChartNext</a>	Возвращает идентификатор графика, следующего за указанным	<a href="#">Операции с графиками</a>
<a href="#">ChartOpen</a>	Открывает новый график с указанным символом и периодом	<a href="#">Операции с графиками</a>

<a href="#">CharToString</a>	Преобразование кода символа в односимвольную строку	<a href="#">Преобразование данных</a>
<a href="#">ChartPeriod</a>	Возвращает значение периода указанного графика	<a href="#">Операции с графиками</a>
<a href="#">ChartPriceOnDropped</a>	Возвращает ценовую координату, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт	<a href="#">Операции с графиками</a>
<a href="#">ChartRedraw</a>	Вызывает принудительную перерисовку указанного графика	<a href="#">Операции с графиками</a>
<a href="#">ChartSaveTemplate</a>	Сохраняет текущие настройки графика в шаблон с указанным именем	<a href="#">Операции с графиками</a>
<a href="#">ChartScreenShot</a>	Делает снимок указанного графика в формате GIF, PNG или BMP в зависимости от указанного расширения	<a href="#">Операции с графиками</a>
<a href="#">ChartSetDouble</a>	Задает значение типа double соответствующего свойства указанного графика	<a href="#">Операции с графиками</a>
<a href="#">ChartSetInteger</a>	Задает значение целочисленного типа (datetime, int, color, bool или char) соответствующего свойства указанного графика	<a href="#">Операции с графиками</a>
<a href="#">ChartSetString</a>	Задает значение типа string соответствующего свойства указанного графика	<a href="#">Операции с графиками</a>
<a href="#">ChartSetSymbolPeriod</a>	Меняет значения символа и периода указанного графика	<a href="#">Операции с графиками</a>
<a href="#">ChartSymbol</a>	Возвращает имя символа указанного графика	<a href="#">Операции с графиками</a>
<a href="#">ChartTimeOnDropped</a>	Возвращает временную координату, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт	<a href="#">Операции с графиками</a>
<a href="#">ChartTimePriceToXY</a>	Преобразует координаты графика из представления	<a href="#">Операции с графиками</a>

	время/цена в координаты по оси X и Y	
<a href="#">ChartWindowFind</a>	Возвращает номер подокна, в котором находится индикатор	<a href="#">Операции с графиками</a>
<a href="#">ChartWindowOnDropped</a>	Возвращает номер подокна графика, на которое брошен мышкой данный эксперт, скрипт, объект или индикатор	<a href="#">Операции с графиками</a>
<a href="#">ChartXOnDropped</a>	Возвращает координату по оси X, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт	<a href="#">Операции с графиками</a>
<a href="#">ChartXYToTimePrice</a>	Преобразует координаты X и Y графика в значения время и цена	<a href="#">Операции с графиками</a>
<a href="#">ChartYOnDropped</a>	Возвращает координату по оси Y, соответствующую точке, в которой брошен мышкой данный эксперт или скрипт	<a href="#">Операции с графиками</a>
<a href="#">CheckPointer</a>	Возвращает тип указателя объекта	<a href="#">Общие функции</a>
<a href="#">CLBufferCreate</a>	Создает буфер OpenCL	<a href="#">Работа с OpenCL</a>
<a href="#">CLBufferFree</a>	Удаляет буфер OpenCL	<a href="#">Работа с OpenCL</a>
<a href="#">CLBufferRead</a>	Читает буфер OpenCL в массив и возвращает количество прочитанных элементов	<a href="#">Работа с OpenCL</a>
<a href="#">CLBufferWrite</a>	Записывает массив в буфер OpenCL и возвращает количество записанных элементов	<a href="#">Работа с OpenCL</a>
<a href="#">CLContextCreate</a>	Создает контекст OpenCL	<a href="#">Работа с OpenCL</a>
<a href="#">CLContextFree</a>	Удаляет контекст OpenCL	<a href="#">Работа с OpenCL</a>
<a href="#">CLExecute</a>	Выполняет OpenCL программу	<a href="#">Работа с OpenCL</a>
<a href="#">CLGetDeviceInfo</a>	Получает свойство устройства из OpenCL драйвера	<a href="#">Работа с OpenCL</a>

<a href="#">CLGetInfoInteger</a>	Возвращает значение целочисленного свойства для OpenCL-объекта или устройства	<a href="#">Работа с OpenCL</a>
<a href="#">CLHandleType</a>	Возвращает тип OpenCL хендла в виде значения из перечисления ENUM_OPENCL_HANDLE_TYPE	<a href="#">Работа с OpenCL</a>
<a href="#">CLKernelCreate</a>	Создает функцию запуска OpenCL	<a href="#">Работа с OpenCL</a>
<a href="#">CLKernelFree</a>	Удаляет функцию запуска OpenCL	<a href="#">Работа с OpenCL</a>
<a href="#">CLProgramCreate</a>	Создает OpenCL программу из исходного кода	<a href="#">Работа с OpenCL</a>
<a href="#">CLProgramFree</a>	Удаляет OpenCL программу	<a href="#">Работа с OpenCL</a>
<a href="#">CLSetKernelArg</a>	Выставляет параметр для функции OpenCL	<a href="#">Работа с OpenCL</a>
<a href="#">CLSetKernelArgMem</a>	Выставляет буфер OpenCL в качестве параметра функции OpenCL	<a href="#">Работа с OpenCL</a>
<a href="#">ColorToARGB</a>	Преобразует тип color в тип uint для получения ARGB-представления цвета.	<a href="#">Преобразование данных</a>
<a href="#">ColorToString</a>	Преобразует значение цвета в строку вида "R,G,B"	<a href="#">Преобразование данных</a>
<a href="#">Comment</a>	Выводит сообщение в левый верхний угол ценового графика	<a href="#">Общие функции</a>
<a href="#">CopyBuffer</a>	Получает в массив данные указанного буфера от указанного индикатора	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">CopyClose</a>	Получает в массив исторические данные по цене закрытия баров по соответствующим символу и периоду	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">CopyHigh</a>	Получает в массив исторические данные по максимальной цене баров по соответствующим символу и периоду	<a href="#">Доступ к таймсериям и индикаторам</a>

<a href="#">CopyLow</a>	Получает в массив исторические данные по минимальной цене баров по соответствующим символу и периоду	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">CopyOpen</a>	Получает в массив исторические данные по цене открытия баров по соответствующим символу и периоду	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">CopyRates</a>	Получает в массив исторические данные структуры <a href="#">Rates</a> для указанных символа и периода	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">CopyRealVolume</a>	Получает в массив исторические данные по торговым объемам для соответствующих символа и периода	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">CopySpread</a>	Получает в массив исторические данные по спредам для соответствующих символа и периода	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">CopyTicks</a>	Получает в массив тики, накопленные терминалом за текущую рабочую сессию	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">CopyTickVolume</a>	Получает в массив исторические данные по тиковым объемам для соответствующих символа и периода	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">CopyTime</a>	Получает в массив исторические данные по времени открытия баров по соответствующим символу и периоду	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">cos</a>	Возвращает косинус числа	<a href="#">Математические функции</a>
<a href="#">CryptDecode</a>	Производит обратное преобразование данных массива	<a href="#">Общие функции</a>
<a href="#">CryptEncode</a>	Преобразует данные массива-источника в массив-приемник	<a href="#">Общие функции</a>

	указанным методом	
<a href="#">CustomSymbolCreate</a>	Создает пользовательский символ с указанным именем в указанной группе	<a href="#">Пользовательские символы</a>
<a href="#">CustomSymbolDelete</a>	Удаляет пользовательский символ с указанным именем	<a href="#">Пользовательские символы</a>
<a href="#">CustomSymbolSetInteger</a>	Устанавливает для пользовательского символа значение свойства целочисленного типа	<a href="#">Пользовательские символы</a>
<a href="#">CustomSymbolSetDouble</a>	Устанавливает для пользовательского символа значение свойства вещественного типа	<a href="#">Пользовательские символы</a>
<a href="#">CustomSymbolSetString</a>	Устанавливает для пользовательского символа значение свойства строкового типа	<a href="#">Пользовательские символы</a>
<a href="#">CustomSymbolSetMarginRate</a>	Устанавливает для пользовательского символа коэффициенты взимания маржи в зависимости от типа и направления ордера	<a href="#">Пользовательские символы</a>
<a href="#">CustomSymbolSetSessionQuote</a>	Устанавливает время начала и время окончания указанной котировочной сессии для указанных символа и дня недели	<a href="#">Пользовательские символы</a>
<a href="#">CustomSymbolSetSessionTrade</a>	Устанавливает время начала и время окончания указанной торговой сессии для указанных символа и дня недели	<a href="#">Пользовательские символы</a>
<a href="#">CustomRatesDelete</a>	Удаляет все бары в указанном временном интервале из ценовой истории пользовательского инструмента	<a href="#">Пользовательские символы</a>
<a href="#">CustomRatesReplace</a>	Полностью заменяет ценовую историю пользовательского инструмента в указанном временном интервале данными из массива типа <code>MqlRates</code>	<a href="#">Пользовательские символы</a>

<a href="#">CustomRatesUpdate</a>	Добавляет в историю пользователя инструмента отсутствующие бары и заменяет существующие бары данными из массива типа <code>MqlRates</code>	<a href="#">Пользовательские символы</a>
<a href="#">CustomTicksAdd</a>	Добавляет в ценовую историю пользователя инструмента данные из массива типа <code>MqlTick</code> . Пользовательский символ должен быть выбран в окне MarketWatch (Обзор рынка)	<a href="#">Пользовательские символы</a>
<a href="#">CustomTicksDelete</a>	Удаляет все тики в указанном временном интервале из ценовой истории пользователя инструмента	<a href="#">Пользовательские символы</a>
<a href="#">CustomTicksReplace</a>	Полностью заменяет ценовую историю пользователя инструмента в указанном временном интервале данными из массива типа <code>MqlTick</code>	<a href="#">Пользовательские символы</a>
<a href="#">CustomBookAdd</a>	Передает состояние стакана цен по пользовательскому инструменту	<a href="#">Пользовательские символы</a>
<a href="#">DebugBreak</a>	Программная точка останова при отладке	<a href="#">Общие функции</a>
<a href="#">Digits</a>	Возвращает количество десятичных знаков после запятой, определяющее точность измерения цены символа текущего графика	<a href="#">Проверка состояния</a>
<a href="#">DoubleToString</a>	Преобразование числового значения в текстовую строку с указанной точностью	<a href="#">Преобразование данных</a>
<a href="#">EnumToString</a>	Преобразование значения перечисления любого типа в строку	<a href="#">Преобразование данных</a>
<a href="#">EventChartCustom</a>	Генерирует пользовательское событие для указанного графика	<a href="#">Работа с событиями</a>

<a href="#">EventKillTimer</a>	Останавливает на текущем графике генерацию событий по таймеру	<a href="#">Работа с событиями</a>
<a href="#">EventSetMillisecondTimer</a>	Запускает генератор событий таймера высокого разрешения с периодом менее 1 секунды для текущего графика	<a href="#">Работа с событиями</a>
<a href="#">EventSetTimer</a>	Запускает генератор событий таймера с указанной периодичностью для текущего графика	<a href="#">Работа с событиями</a>
<a href="#">exp</a>	Возвращает экспоненту числа	<a href="#">Математические функции</a>
<a href="#">ExpertRemove</a>	Прекращает работу эксперта и выгружает его с графика	<a href="#">Общие функции</a>
<a href="#">fabs</a>	Возвращает абсолютное значение (значение по модулю) переданного ей числа	<a href="#">Математические функции</a>
<a href="#">FileClose</a>	Закрывает ранее открытый файл	<a href="#">Файловые операции</a>
<a href="#">FileCopy</a>	Копирует исходный файл из локальной или общей папки в другой файл	<a href="#">Файловые операции</a>
<a href="#">FileDelete</a>	Удаляет указанный файл	<a href="#">Файловые операции</a>
<a href="#">FileFindClose</a>	Закрывает хэндл поиска	<a href="#">Файловые операции</a>
<a href="#">FileFindFirst</a>	Начинает перебор файлов в соответствующей директории в соответствии с указанным фильтром	<a href="#">Файловые операции</a>
<a href="#">FileFindNext</a>	Продолжает поиск, начатый функцией FileFindFirst()	<a href="#">Файловые операции</a>
<a href="#">FileFlush</a>	Сброс на диск всех данных, оставшихся в файловом буфере ввода-вывода	<a href="#">Файловые операции</a>
<a href="#">FileGetInteger</a>	Получает целочисленное свойство файла	<a href="#">Файловые операции</a>
<a href="#">FileIsEnding</a>	Определяет конец файла в процессе чтения	<a href="#">Файловые операции</a>
<a href="#">FileIsExist</a>	Проверяет существование файла	<a href="#">Файловые операции</a>

<a href="#">FileIsLineEnding</a>	Определяет конец строки в текстовом файле в процессе чтения	<a href="#">Файловые операции</a>
<a href="#">FileMove</a>	Перемещает или переименовывает файл	<a href="#">Файловые операции</a>
<a href="#">FileOpen</a>	Открывает файл с указанным именем и указанными флагами	<a href="#">Файловые операции</a>
<a href="#">FileReadArray</a>	Читает массивы любых типов, кроме строковых (может быть массив структур, не содержащих строки и динамические массивы), из бинарного файла с текущего положения файлового указателя	<a href="#">Файловые операции</a>
<a href="#">FileReadBool</a>	Читает из файла типа CSV строку от текущего положения до разделителя (либо до конца текстовой строки) и преобразует прочитанную строку в значение типа bool	<a href="#">Файловые операции</a>
<a href="#">FileReadDatetime</a>	Читает из файла типа CSV строку одного из форматов: "YYYY.MM.DD HH:MM:SS", "YYYY.MM.DD" или "HH:MM:SS" - и преобразует ее в значение типа datetime	<a href="#">Файловые операции</a>
<a href="#">FileReadDouble</a>	Читает число двойной точности с плавающей точкой (double) из бинарного файла с текущего положения файлового указателя	<a href="#">Файловые операции</a>
<a href="#">FileReadFloat</a>	Читает из текущего положения файлового указателя значение типа float	<a href="#">Файловые операции</a>
<a href="#">FileReadInteger</a>	Читает из бинарного файла значение типа int, short или char в зависимости от указанной длины в байтах	<a href="#">Файловые операции</a>
<a href="#">FileReadLong</a>	Читает из текущего положения файлового указателя значение типа long	<a href="#">Файловые операции</a>

<a href="#">FileReadNumber</a>	Читает из файла типа CSV строку от текущего положения до разделителя (либо до конца текстовой строки) и преобразует прочитанную строку в значение типа double	<a href="#">Файловые операции</a>
<a href="#">FileReadString</a>	Читает из файла строку с текущего положения файлового указателя	<a href="#">Файловые операции</a>
<a href="#">FileReadStruct</a>	Считывает из бинарного файла содержимое в структуру, переданную в качестве параметра	<a href="#">Файловые операции</a>
<a href="#">FileSeek</a>	Перемещает положение файлового указателя на указанное количество байт относительно указанного положения	<a href="#">Файловые операции</a>
<a href="#">FileSize</a>	Возвращает размер соответствующего открытого файла	<a href="#">Файловые операции</a>
<a href="#">FileTell</a>	Возвращает текущее положение файлового указателя соответствующего открытого файла	<a href="#">Файловые операции</a>
<a href="#">FileWrite</a>	Записывает данные в файл типа CSV или TXT	<a href="#">Файловые операции</a>
<a href="#">FileWriteArray</a>	Записывает в файл типа BIN массивы любых типов, кроме строковых	<a href="#">Файловые операции</a>
<a href="#">FileWriteDouble</a>	Записывает в двоичный файл значение параметра типа double с текущего положения файлового указателя	<a href="#">Файловые операции</a>
<a href="#">FileWriteFloat</a>	Записывает в двоичный файл значение параметра типа float с текущего положения файлового указателя	<a href="#">Файловые операции</a>
<a href="#">FileWriteInteger</a>	Записывает в двоичный файл значение параметра типа int с текущего положения файлового указателя	<a href="#">Файловые операции</a>

<a href="#">FileWriteLong</a>	Записывает в двоичный файл значение параметра типа long с текущего положения файлового указателя	<a href="#">Файловые операции</a>
<a href="#">FileWriteString</a>	Записывает в файл типа BIN или TXT значение параметра типа string с текущего положения файлового указателя	<a href="#">Файловые операции</a>
<a href="#">FileWriteStruct</a>	Записывает в двоичный файл содержимое структуры, переданной в качестве параметра, с текущего положения файлового указателя	<a href="#">Файловые операции</a>
<a href="#">floor</a>	Возвращает ближайшее снизу целое числовое значение	<a href="#">Математические функции</a>
<a href="#">fmax</a>	Возвращает максимальное из двух числовых значений	<a href="#">Математические функции</a>
<a href="#">fmin</a>	Возвращает минимальное из двух числовых значений	<a href="#">Математические функции</a>
<a href="#">fmod</a>	Возвращает вещественный остаток от деления двух чисел	<a href="#">Математические функции</a>
<a href="#">FolderClean</a>	Удаляет все файлы в указанной папке	<a href="#">Файловые операции</a>
<a href="#">FolderCreate</a>	Создает директорию в папке Files (в зависимости от значения common_flag)	<a href="#">Файловые операции</a>
<a href="#">FolderDelete</a>	Удаляет указанную директорию. Если папка не пуста, то она не может быть удалена	<a href="#">Файловые операции</a>
<a href="#">FrameAdd</a>	Добавляет фрейм с данными	<a href="#">Работа с результатами оптимизации</a>
<a href="#">FrameFilter</a>	Устанавливает фильтр чтения фреймов и переводит указатель на начало	<a href="#">Работа с результатами оптимизации</a>
<a href="#">FrameFirst</a>	Переводит указатель чтения фреймов в начало и сбрасывает ранее установленный фильтр	<a href="#">Работа с результатами оптимизации</a>

<a href="#">FrameInputs</a>	Получает <a href="#">input-параметры</a> , на которых сформирован фрейм	<a href="#">Работа с результатами оптимизации</a>
<a href="#">FrameNext</a>	Читает фрейм и перемещает указатель на следующий	<a href="#">Работа с результатами оптимизации</a>
<a href="#">GetLastError</a>	Возвращает значение последней ошибки	<a href="#">Проверка состояния</a>
<a href="#">GetPointer</a>	Возвращает <a href="#">указатель</a> объекта	<a href="#">Общие функции</a>
<a href="#">GetTickCount</a>	Возвращает количество миллисекунд, прошедших с момента старта системы	<a href="#">Общие функции</a>
<a href="#">GlobalVariableCheck</a>	Проверяет существование глобальной переменной с указанным именем	<a href="#">Глобальные переменные терминала</a>
<a href="#">GlobalVariableDel</a>	Удаляет глобальную переменную	<a href="#">Глобальные переменные терминала</a>
<a href="#">GlobalVariableGet</a>	Запрашивает значение глобальной переменной	<a href="#">Глобальные переменные терминала</a>
<a href="#">GlobalVariableName</a>	Возвращает имя глобальной переменной по порядковому номеру в списке глобальных переменных	<a href="#">Глобальные переменные терминала</a>
<a href="#">GlobalVariablesDeleteAll</a>	Удаляет глобальные переменные с указанным префиксом в имени	<a href="#">Глобальные переменные терминала</a>
<a href="#">GlobalVariableSet</a>	Устанавливает новое значение глобальной переменной	<a href="#">Глобальные переменные терминала</a>
<a href="#">GlobalVariableSetOnCondition</a>	Устанавливает новое значение существующей глобальной переменной по условию	<a href="#">Глобальные переменные терминала</a>
<a href="#">GlobalVariablesFlush</a>	Принудительно записывает содержимое всех глобальных переменных на диск	<a href="#">Глобальные переменные терминала</a>
<a href="#">GlobalVariablesTotal</a>	Возвращает общее количество глобальных переменных	<a href="#">Глобальные переменные терминала</a>
<a href="#">GlobalVariableTemp</a>	Устанавливает новое значение глобальной переменной, которая	<a href="#">Глобальные переменные терминала</a>

	существует только на время текущего сеанса работы терминала	
<a href="#">GlobalVariableTime</a>	Возвращает время последнего доступа к глобальной переменной	<a href="#">Глобальные переменные терминала</a>
<a href="#">HistoryDealGetDouble</a>	Возвращает запрошенное свойство сделки в истории (double)	<a href="#">Торговые функции</a>
<a href="#">HistoryDealGetInteger</a>	Возвращает запрошенное свойство сделки в истории (datetime или int)	<a href="#">Торговые функции</a>
<a href="#">HistoryDealGetString</a>	Возвращает запрошенное свойство сделки в истории (string)	<a href="#">Торговые функции</a>
<a href="#">HistoryDealGetTicket</a>	Выбирает сделку для дальнейшей обработки и возвращает тикет сделки в истории	<a href="#">Торговые функции</a>
<a href="#">HistoryDealSelect</a>	Выбирает в истории сделку для дальнейших обращений к ней через соответствующие функции	<a href="#">Торговые функции</a>
<a href="#">HistoryDealsTotal</a>	Возвращает количество сделок в истории	<a href="#">Торговые функции</a>
<a href="#">HistoryOrderGetDouble</a>	Возвращает запрошенное свойство ордера в истории (double)	<a href="#">Торговые функции</a>
<a href="#">HistoryOrderGetInteger</a>	Возвращает запрошенное свойство ордера в истории (datetime или int)	<a href="#">Торговые функции</a>
<a href="#">HistoryOrderGetString</a>	Возвращает запрошенное свойство ордера в истории (string)	<a href="#">Торговые функции</a>
<a href="#">HistoryOrderGetTicket</a>	Возвращает тикет соответствующего ордера в истории	<a href="#">Торговые функции</a>
<a href="#">HistoryOrderSelect</a>	Выбирает в истории ордер для дальнейшей работы с ним	<a href="#">Торговые функции</a>
<a href="#">HistoryOrdersTotal</a>	Возвращает количество ордеров в истории	<a href="#">Торговые функции</a>

<a href="#">HistorySelect</a>	Запрашивает историю сделок и ордеров за указанный период серверного времени	<a href="#">Торговые функции</a>
<a href="#">HistorySelectByPosition</a>	Запрашивает историю сделок и ордеров с указанным <a href="#">идентификатором позиции</a> .	<a href="#">Торговые функции</a>
<a href="#">iAC</a>	Accelerator Oscillator	<a href="#">Технические индикаторы</a>
<a href="#">iAD</a>	Accumulation/Distribution	<a href="#">Технические индикаторы</a>
<a href="#">iADX</a>	Average Directional Index	<a href="#">Технические индикаторы</a>
<a href="#">iADXWilder</a>	Average Directional Index by Welles Wilder	<a href="#">Технические индикаторы</a>
<a href="#">iAlligator</a>	Alligator	<a href="#">Технические индикаторы</a>
<a href="#">iAMA</a>	Adaptive Moving Average	<a href="#">Технические индикаторы</a>
<a href="#">iAO</a>	Awesome Oscillator	<a href="#">Технические индикаторы</a>
<a href="#">iATR</a>	Average True Range	<a href="#">Технические индикаторы</a>
<a href="#">iBands</a>	Bollinger Bands®	<a href="#">Технические индикаторы</a>
<a href="#">iBearsPower</a>	Bears Power	<a href="#">Технические индикаторы</a>
<a href="#">iBullsPower</a>	Bulls Power	<a href="#">Технические индикаторы</a>
<a href="#">iBWMFI</a>	Market Facilitation Index by Bill Williams	<a href="#">Технические индикаторы</a>
<a href="#">iCCI</a>	Commodity Channel Index	<a href="#">Технические индикаторы</a>
<a href="#">iChaikin</a>	Chaikin Oscillator	<a href="#">Технические индикаторы</a>
<a href="#">iCustom</a>	пользовательский индикатор	<a href="#">Технические индикаторы</a>
<a href="#">iDEMA</a>	Double Exponential Moving Average	<a href="#">Технические индикаторы</a>
<a href="#">iDeMarker</a>	DeMarker	<a href="#">Технические индикаторы</a>
<a href="#">iEnvelopes</a>	Envelopes	<a href="#">Технические индикаторы</a>
<a href="#">iForce</a>	Force Index	<a href="#">Технические индикаторы</a>
<a href="#">iFractals</a>	Fractals	<a href="#">Технические индикаторы</a>
<a href="#">iFrAMA</a>	Fractal Adaptive Moving Average	<a href="#">Технические индикаторы</a>
<a href="#">iGator</a>	Gator Oscillator	<a href="#">Технические индикаторы</a>
<a href="#">iIchimoku</a>	Ichimoku Kinko Hyo	<a href="#">Технические индикаторы</a>
<a href="#">iMA</a>	Moving Average	<a href="#">Технические индикаторы</a>

<a href="#">iMACD</a>	Moving Averages Convergence-Divergence	<a href="#">Технические индикаторы</a>
<a href="#">iMFI</a>	Money Flow Index	<a href="#">Технические индикаторы</a>
<a href="#">iMomentum</a>	Momentum	<a href="#">Технические индикаторы</a>
<a href="#">IndicatorCreate</a>	Возвращает хэндл указанного технического индикатора, созданного на основе массива параметров типа <a href="#">MqlParam</a>	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">IndicatorParameters</a>	Возвращает по указанному хэндулу количество входных параметров индикатора, а также сами значения и тип параметров	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">IndicatorRelease</a>	Удаляет хэндл индикатора и освобождает расчетную часть индикатора, если ею больше никто не пользуется	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">IndicatorSetDouble</a>	Задает значение свойства индикатора, имеющего тип <a href="#">double</a>	<a href="#">Пользовательские индикаторы</a>
<a href="#">IndicatorSetInteger</a>	Задает значение свойства индикатора, имеющего тип <a href="#">int</a>	<a href="#">Пользовательские индикаторы</a>
<a href="#">IndicatorSetString</a>	Задает значение свойства индикатора, имеющего тип <a href="#">string</a>	<a href="#">Пользовательские индикаторы</a>
<a href="#">IntegerToString</a>	Преобразование значения целого типа в строку указанной длины	<a href="#">Преобразование данных</a>
<a href="#">iOBV</a>	On Balance Volume	<a href="#">Технические индикаторы</a>
<a href="#">iOsMA</a>	Moving Average of Oscillator (MACD histogram)	<a href="#">Технические индикаторы</a>
<a href="#">iRSI</a>	Relative Strength Index	<a href="#">Технические индикаторы</a>
<a href="#">iRVI</a>	Relative Vigor Index	<a href="#">Технические индикаторы</a>
<a href="#">iSAR</a>	Parabolic Stop And Reverse System	<a href="#">Технические индикаторы</a>
<a href="#">IsStopped</a>	Возвращает true, если поступила команда завершить выполнение mql5-программы	<a href="#">Проверка состояния</a>

<a href="#">iStdDev</a>	Standard Deviation	<a href="#">Технические индикаторы</a>
<a href="#">iStochastic</a>	Stochastic Oscillator	<a href="#">Технические индикаторы</a>
<a href="#">iTEMA</a>	Triple Exponential Moving Average	<a href="#">Технические индикаторы</a>
<a href="#">iTriX</a>	Triple Exponential Moving Averages Oscillator	<a href="#">Технические индикаторы</a>
<a href="#">iVIDyA</a>	Variable Index Dynamic Average	<a href="#">Технические индикаторы</a>
<a href="#">iVolumes</a>	Volumes	<a href="#">Технические индикаторы</a>
<a href="#">iWPR</a>	Williams' Percent Range	<a href="#">Технические индикаторы</a>
<a href="#">log</a>	Возвращает натуральный логарифм	<a href="#">Математические функции</a>
<a href="#">log10</a>	Возвращает логарифм числа по основанию 10	<a href="#">Математические функции</a>
<a href="#">MarketBookAdd</a>	Обеспечивает открытие стакана цен по указанному инструменту, а также производит подписку на получение извещений об изменении указанного стакана	<a href="#">Получение рыночной информации</a>
<a href="#">MarketBookGet</a>	Возвращает массив структур типа <a href="#">MqlBookInfo</a> , содержащий записи стакана цен указанного символа	<a href="#">Получение рыночной информации</a>
<a href="#">MarketBookRelease</a>	Обеспечивает закрытие стакана цен по указанному инструменту, а также отменяет подписку на получение извещений об изменении указанного стакана	<a href="#">Получение рыночной информации</a>
<a href="#">MathAbs</a>	Возвращает абсолютное значение (значение по модулю) переданного ей числа	<a href="#">Математические функции</a>
<a href="#">MathArccos</a>	Возвращает значение арккосинуса x в радианах	<a href="#">Математические функции</a>
<a href="#">MathArcsin</a>	Возвращает значение арксинуса x в радианах	<a href="#">Математические функции</a>

<a href="#">MathArctan</a>	Возвращает арктангенс $x$ в радианах	<a href="#">Математические функции</a>
<a href="#">MathCeil</a>	Возвращает ближайшее сверху целое числовое значение	<a href="#">Математические функции</a>
<a href="#">MathCos</a>	Возвращает косинус числа	<a href="#">Математические функции</a>
<a href="#">MathExp</a>	Возвращает экспоненту числа	<a href="#">Математические функции</a>
<a href="#">MathFloor</a>	Возвращает ближайшее снизу целое числовое значение	<a href="#">Математические функции</a>
<a href="#">MathIsValidNumber</a>	Проверяет корректность действительного числа	<a href="#">Математические функции</a>
<a href="#">MathLog</a>	Возвращает натуральный логарифм	<a href="#">Математические функции</a>
<a href="#">MathLog10</a>	Возвращает логарифм числа по основанию 10	<a href="#">Математические функции</a>
<a href="#">MathMax</a>	Возвращает максимальное из двух числовых значений	<a href="#">Математические функции</a>
<a href="#">MathMin</a>	Возвращает минимальное из двух числовых значений	<a href="#">Математические функции</a>
<a href="#">MathMod</a>	Возвращает вещественный остаток от деления двух чисел	<a href="#">Математические функции</a>
<a href="#">MathPow</a>	Возводит основание в указанную степень	<a href="#">Математические функции</a>
<a href="#">MathRand</a>	Возвращает псевдослучайное целое число в диапазоне от 0 до 32767	<a href="#">Математические функции</a>
<a href="#">MathRound</a>	Округляет число до ближайшего целого	<a href="#">Математические функции</a>
<a href="#">MathSin</a>	Возвращает синус числа	<a href="#">Математические функции</a>
<a href="#">MathSqrt</a>	Возвращает квадратный корень	<a href="#">Математические функции</a>
<a href="#">MathSrand</a>	Устанавливает начальное состояние генератора псевдослучайных целых чисел	<a href="#">Математические функции</a>
<a href="#">MathTan</a>	Возвращает тангенс числа	<a href="#">Математические функции</a>
<a href="#">MessageBox</a>	Создает и отображает окно сообщений, а также	<a href="#">Общие функции</a>

	управляет им	
<a href="#">MQLInfoInteger</a>	Возвращает значение целого типа соответствующего свойства запущенной mql5-программы	<a href="#">Проверка состояния</a>
<a href="#">MQLInfoString</a>	Возвращает значение типа string соответствующего свойства запущенной mql5-программы	<a href="#">Проверка состояния</a>
<a href="#">MT5Initialize</a>	Устанавливает соединение с терминалом MetaTrader 5	<a href="#">MetaTrader для Python</a>
<a href="#">MT5Shutdown</a>	Закрывает ранее установленное подключение к терминалу MetaTrader 5	<a href="#">MetaTrader для Python</a>
<a href="#">MT5TerminalInfo</a>	Получает состояние и параметры подключенного терминала MetaTrader 5	<a href="#">MetaTrader для Python</a>
<a href="#">MT5Version</a>	Возвращает версию терминала MetaTrader 5	<a href="#">MetaTrader для Python</a>
<a href="#">MT5WaitForTerminal</a>	Ждет пока терминал MetaTrader 5 подключится к торговому серверу	<a href="#">MetaTrader для Python</a>
<a href="#">MT5CopyRatesFrom</a>	Получает бары из терминала MetaTrader 5, начиная с указанной даты	<a href="#">MetaTrader для Python</a>
<a href="#">MT5CopyRatesFromPos</a>	Получает бары из терминала MetaTrader 5, начиная с указанного индекса	<a href="#">MetaTrader для Python</a>
<a href="#">MT5CopyRatesRange</a>	Получает бары в указанном диапазоне дат из терминала MetaTrader 5	<a href="#">MetaTrader для Python</a>
<a href="#">MT5CopyTicksFrom</a>	Получает тики из терминала MetaTrader 5, начиная с указанной даты	<a href="#">MetaTrader для Python</a>
<a href="#">MT5CopyTicksRange</a>	Получает тики за указанный диапазон дат из терминала MetaTrader 5	<a href="#">MetaTrader для Python</a>
<a href="#">NormalizeDouble</a>	Округление числа с плавающей точкой до указанной точности	<a href="#">Преобразование данных</a>
<a href="#">ObjectCreate</a>	Создает объект заданного типа на указанном графике	<a href="#">Графические объекты</a>

<a href="#">ObjectDelete</a>	Удаляет объект с указанным именем с указанного графика (с указанного подокна графика)	<a href="#">Графические объекты</a>
<a href="#">ObjectFind</a>	Ищет по имени объект с указанным идентификатором	<a href="#">Графические объекты</a>
<a href="#">ObjectGetDouble</a>	Возвращает значение типа double соответствующего свойства объекта	<a href="#">Графические объекты</a>
<a href="#">ObjectGetInteger</a>	Возвращает целочисленное значение соответствующего свойства объекта	<a href="#">Графические объекты</a>
<a href="#">ObjectGetString</a>	Возвращает значение типа string соответствующего свойства объекта	<a href="#">Графические объекты</a>
<a href="#">ObjectGetTimeByValue</a>	Возвращает значение времени для указанного значения цены объекта	<a href="#">Графические объекты</a>
<a href="#">ObjectGetValueByTime</a>	Возвращает ценовое значение объекта для указанного времени	<a href="#">Графические объекты</a>
<a href="#">ObjectMove</a>	Изменяет координаты указанной точки привязки объекта	<a href="#">Графические объекты</a>
<a href="#">ObjectName</a>	Возвращает имя объекта соответствующего типа в указанном графике (указанном подокне графика)	<a href="#">Графические объекты</a>
<a href="#">ObjectsDeleteAll</a>	Удаляет все объекты указанного типа с указанного графика (с указанного подокна графика)	<a href="#">Графические объекты</a>
<a href="#">ObjectSetDouble</a>	Устанавливает значение соответствующего свойства объекта	<a href="#">Графические объекты</a>
<a href="#">ObjectSetInteger</a>	Устанавливает значение соответствующего свойства объекта	<a href="#">Графические объекты</a>
<a href="#">ObjectSetString</a>	Устанавливает значение соответствующего свойства объекта	<a href="#">Графические объекты</a>

<a href="#"><u>ObjectsTotal</u></a>	Возвращает количество объектов указанного типа в указанном графике (указанном подокне графика)	<a href="#"><u>Графические объекты</u></a>
<a href="#"><u>OnStart</u></a>	Вызывается в скрипте при наступлении события <a href="#"><u>Start</u></a> для выполнения действий, заложенных в скрипт	<a href="#"><u>Обработка событий</u></a>
<a href="#"><u>OnInit</u></a>	Вызывается в индикаторах и экспертах при наступлении события <a href="#"><u>Init</u></a> для инициализации запущенной MQL5-программы	<a href="#"><u>Обработка событий</u></a>
<a href="#"><u>OnDeinit</u></a>	Вызывается в индикаторах и экспертах при наступлении события <a href="#"><u>Deinit</u></a> для деинициализации запущенной MQL5-программы	<a href="#"><u>Обработка событий</u></a>
<a href="#"><u>OnTick</u></a>	Вызывается в экспертах при наступлении события <a href="#"><u>NewTick</u></a> для обработки новой котировки	<a href="#"><u>Обработка событий</u></a>
<a href="#"><u>OnCalculate</u></a>	Вызывается в индикаторах при наступлении события <a href="#"><u>Calculate</u></a> для обработки изменения ценовых данных	<a href="#"><u>Обработка событий</u></a>
<a href="#"><u>OnTimer</u></a>	Вызывается в индикаторах и экспертах при наступлении периодического события <a href="#"><u>Timer</u></a> , которое с заданным интервалом времени генерируется терминалом	<a href="#"><u>Обработка событий</u></a>
<a href="#"><u>OnTrade</u></a>	Вызывается в экспертах при наступлении события <a href="#"><u>Trade</u></a> , которое генерируется при завершении торговой операции на торговом сервере	<a href="#"><u>Обработка событий</u></a>
<a href="#"><u>OnTradeTransaction</u></a>	Вызывается в экспертах при наступлении события <a href="#"><u>TradeTransaction</u></a> для обработки результатов выполнения торгового запроса	<a href="#"><u>Обработка событий</u></a>

<a href="#">OnBookEvent</a>	Вызывается в экспертах при наступлении события <a href="#">BookEvent</a> для обработки изменений в стакане заявок	<a href="#">Обработка событий</a>
<a href="#">OnChartEvent</a>	Вызывается в индикаторах и экспертах при наступлении события <a href="#">ChartEvent</a> для обработки изменений графика, вызванных действиями пользователя или работой MQL5-программ	<a href="#">Обработка событий</a>
<a href="#">OnTester</a>	Вызывается в экспертах при наступлении события <a href="#">Tester</a> для выполнения необходимых действий по окончании тестирования	<a href="#">Обработка событий</a>
<a href="#">OnTesterInit</a>	Вызывается в экспертах при наступлении события <a href="#">TesterInit</a> для выполнения необходимых действий перед началом оптимизации	<a href="#">Обработка событий</a>
<a href="#">OnTesterDeinit</a>	Вызывается в экспертах при наступлении события <a href="#">TesterDeinit</a> для выполнения необходимых действий по окончании оптимизации эксперта	<a href="#">Обработка событий</a>
<a href="#">OnTesterPass</a>	Вызывается в экспертах при наступлении события <a href="#">TesterPass</a> для обработки поступления нового фрейма данных во время оптимизации эксперта	<a href="#">Обработка событий</a>
<a href="#">OrderCalcMargin</a>	Вычисляет размер маржи, необходимой для указанного типа ордера, в валюте счета	<a href="#">Торговые функции</a>
<a href="#">OrderCalcProfit</a>	Вычисляет размер прибыли на основании переданных параметров в валюте счета	<a href="#">Торговые функции</a>
<a href="#">OrderCheck</a>	Проверяет достаточность средств для совершения требуемой <a href="#">торговой операции</a> .	<a href="#">Торговые функции</a>
<a href="#">OrderGetDouble</a>	Возвращает запрошенное свойство ордера (double)	<a href="#">Торговые функции</a>

<a href="#">OrderGetInteger</a>	Возвращает запрошенное свойство ордера (datetime или int)	<a href="#">Торговые функции</a>
<a href="#">OrderGetString</a>	Возвращает запрошенное свойство ордера (string)	<a href="#">Торговые функции</a>
<a href="#">OrderGetTicket</a>	Возвращает тикет соответствующего ордера	<a href="#">Торговые функции</a>
<a href="#">OrderSelect</a>	Выбирает ордер для дальнейшей работы с ним	<a href="#">Торговые функции</a>
<a href="#">OrderSend</a>	Отправляет <a href="#">торговые запросы</a> на сервер	<a href="#">Торговые функции</a>
<a href="#">OrderSendAsync</a>	Отправляет асинхронно <a href="#">торговые запросы</a> без ожидания ответа торгового сервера	<a href="#">Торговые функции</a>
<a href="#">OrdersTotal</a>	Возвращает количество ордеров	<a href="#">Торговые функции</a>
<a href="#">ParameterGetRange</a>	Получает для <a href="#">input-переменной</a> информацию о диапазоне значений и шаге изменения при оптимизации эксперта в тестере стратегий	<a href="#">Работа с результатами оптимизации</a>
<a href="#">ParameterSetRange</a>	Устанавливает правила использования <a href="#">input-переменной</a> при оптимизации эксперта в тестере стратегий: значение, шаг изменения, начальное и конечное значения	<a href="#">Работа с результатами оптимизации</a>
<a href="#">Period</a>	Возвращает значение таймфрейма текущего графика	<a href="#">Проверка состояния</a>
<a href="#">PeriodSeconds</a>	Возвращает количество секунд в периоде	<a href="#">Общие функции</a>
<a href="#">PlaySound</a>	Воспроизводит звуковой файл	<a href="#">Общие функции</a>
<a href="#">PlotIndexGetInteger</a>	Возвращает значение свойства линии индикатора, имеющего <a href="#">целый</a> тип	<a href="#">Пользовательские индикаторы</a>
<a href="#">PlotIndexSetDouble</a>	Задает значение свойства линии индикатора, имеющего тип <a href="#">double</a>	<a href="#">Пользовательские индикаторы</a>

<a href="#">PlotIndexSetInteger</a>	Задает значение свойства линии индикатора, имеющего тип <a href="#">int</a>	<a href="#">Пользовательские индикаторы</a>
<a href="#">PlotIndexSetString</a>	Задает значение свойства линии индикатора, имеющего тип <a href="#">string</a>	<a href="#">Пользовательские индикаторы</a>
<a href="#">Point</a>	Возвращает размер пункта текущего инструмента в валюте котировки.	<a href="#">Проверка состояния</a>
<a href="#">PositionGetDouble</a>	Возвращает запрошенное свойство открытой позиции (double)	<a href="#">Торговые функции</a>
<a href="#">PositionGetInteger</a>	Возвращает запрошенное свойство открытой позиции (datetime или int)	<a href="#">Торговые функции</a>
<a href="#">PositionGetString</a>	Возвращает запрошенное свойство открытой позиции (string)	<a href="#">Торговые функции</a>
<a href="#">PositionGetSymbol</a>	Возвращает символ соответствующей открытой позиции	<a href="#">Торговые функции</a>
<a href="#">PositionGetTicket</a>	Возвращает тикет позиции по индексу в списке открытых позиций	<a href="#">Торговые функции</a>
<a href="#">PositionSelect</a>	Выбирает открытую позицию для дальнейшей работы с ней	<a href="#">Торговые функции</a>
<a href="#">PositionSelectByTicket</a>	Выбирает открытую позицию для дальнейшей работы с ней по указанному тикету	<a href="#">Торговые функции</a>
<a href="#">PositionsTotal</a>	Возвращает количество открытых позиций	<a href="#">Торговые функции</a>
<a href="#">pow</a>	Возводит основание в указанную степень	<a href="#">Математические функции</a>
<a href="#">Print</a>	Выводит сообщение в журнал	<a href="#">Общие функции</a>
<a href="#">PrintFormat</a>	Форматирует и печатает наборы символов и значений в лог-файл в соответствие с заданным форматом	<a href="#">Общие функции</a>
<a href="#">rand</a>	Возвращает псевдослучайное целое число в диапазоне от 0 до 32767	<a href="#">Математические функции</a>

<a href="#">ResetLastError</a>	Устанавливает значение предопределенной переменной <a href="#">LastError</a> в ноль	<a href="#">Общие функции</a>
<a href="#">ResourceCreate</a>	Создает ресурс изображения на основе набора данных	<a href="#">Общие функции</a>
<a href="#">ResourceFree</a>	Удаляет <a href="#">динамически созданный</a> <a href="#">ресурс</a> (освобождает занятую ресурсом память)	<a href="#">Общие функции</a>
<a href="#">ResourceReadImage</a>	Читает данные графического ресурса, <a href="#">созданного функцией ResourceCreate()</a> или <a href="#">сохраненного в EX5-файл при компиляции</a>	<a href="#">Общие функции</a>
<a href="#">ResourceSave</a>	Сохраняет ресурс в указанный файл	<a href="#">Общие функции</a>
<a href="#">round</a>	Округляет число до ближайшего целого	<a href="#">Математические функции</a>
<a href="#">SendFTP</a>	Посыпает файл по адресу, указанному в окне настроек на закладке "FTP"	<a href="#">Общие функции</a>
<a href="#">SendMail</a>	Посыпает электронное письмо по адресу, указанному в окне настроек на закладке "Почта"	<a href="#">Общие функции</a>
<a href="#">SendNotification</a>	Посыпает Push-уведомления в мобильные терминалы, чьи MetaQuotes ID указаны на закладке "Уведомления"	<a href="#">Общие функции</a>
<a href="#">SeriesInfoInteger</a>	Возвращает информацию о состоянии исторических данных	<a href="#">Доступ к таймсериям и индикаторам</a>
<a href="#">SetIndexBuffer</a>	Связывает указанный индикаторный буфер с одномерным динамическим массивом типа <a href="#">double</a>	<a href="#">Пользовательские индикаторы</a>
<a href="#">ShortArrayToString</a>	Копирует часть массива в строку	<a href="#">Преобразование данных</a>
<a href="#">ShortToString</a>	Преобразование код символа (unicode) в односимвольную строку	<a href="#">Преобразование данных</a>

<a href="#">SignalBaseGetDouble</a>	Возвращает значение свойства типа double для выбранного сигнала	<a href="#">Управление сигналами</a>
<a href="#">SignalBaseGetInteger</a>	Возвращает значение свойства типа integer для выбранного сигнала	<a href="#">Управление сигналами</a>
<a href="#">SignalBaseGetString</a>	Возвращает значение свойства типа string для выбранного сигнала	<a href="#">Управление сигналами</a>
<a href="#">SignalBaseSelect</a>	Выбирает для работы сигнал из базы торговых сигналов, доступных в терминале	<a href="#">Управление сигналами</a>
<a href="#">SignalBaseTotal</a>	Возвращает общее количество сигналов, доступных в терминале	<a href="#">Управление сигналами</a>
<a href="#">SignalInfoGetDouble</a>	Возвращает из настроек копирования торгового сигнала значение свойства типа double	<a href="#">Управление сигналами</a>
<a href="#">SignalInfoGetInteger</a>	Возвращает из настроек копирования торгового сигнала значение свойства типа integer	<a href="#">Управление сигналами</a>
<a href="#">SignalInfoGetString</a>	Возвращает из настроек копирования торгового сигнала значение свойства типа string	<a href="#">Управление сигналами</a>
<a href="#">SignalInfoSetDouble</a>	Устанавливает в настройках копирования торгового сигнала значение свойства типа double	<a href="#">Управление сигналами</a>
<a href="#">SignalInfoSetInteger</a>	Устанавливает в настройках копирования торгового сигнала значение свойства типа integer	<a href="#">Управление сигналами</a>
<a href="#">SignalSubscribe</a>	Производит подписку на копирование торгового сигнала	<a href="#">Управление сигналами</a>
<a href="#">SignalUnsubscribe</a>	Отменяет подписку на копирование торгового сигнала	<a href="#">Управление сигналами</a>
<a href="#">sin</a>	Возвращает синус числа	<a href="#">Математические функции</a>

<a href="#"><u>Sleep</u></a>	Задерживает выполнение текущего эксперта или скрипта на определенный интервал	<a href="#"><u>Общие функции</u></a>
<a href="#"><u>SocketCreate</u></a>	Создает сокет с указанными флагами и возвращает его хэндл	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketClose</u></a>	Закрывает сокет	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketConnect</u></a>	Выполняет подключение к серверу с контролем таймаута	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketIsConnected</u></a>	Проверяет, подключен ли сокет в текущий момент времени	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketIsReadable</u></a>	Получает количество байт, которое можно прочитать из сокета	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketIsWritable</u></a>	Проверяет, возможна ли запись данных в сокет в текущий момент времени	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketTimeouts</u></a>	Устанавливает таймауты получения и отправки данных для системного объекта сокета	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketRead</u></a>	Читает данные из сокета	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketSend</u></a>	Записывает данные в сокете	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketTlsHandshake</u></a>	Инициирует защищенное TLS (SSL)-соединение с указанным хостом по протоколу TLS Handshake	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketTlsCertificate</u></a>	Получает данные о сертификате, используемом для защиты сетевого соединения	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketTlsRead</u></a>	Читает данные из защищенного TLS-соединения	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketTlsReadAvailable</u></a>	Читает все доступные данные из защищенного TLS-соединения	<a href="#"><u>Сетевые функции</u></a>
<a href="#"><u>SocketTlsSend</u></a>	Отправляет данные через защищенное TLS-соединение	<a href="#"><u>Сетевые функции</u></a>

<a href="#">sqrt</a>	Возвращает квадратный корень	<a href="#">Математические функции</a>
<a href="#">srand</a>	Устанавливает начальное состояние генератора псевдослучайных целых чисел	<a href="#">Математические функции</a>
<a href="#">StringAdd</a>	Присоединяет к концу строки по месту указанную подстроку	<a href="#">Строковые функции</a>
<a href="#">StringBufferLen</a>	Возвращает размер буфера, распределенного для строки	<a href="#">Строковые функции</a>
<a href="#">StringCompare</a>	Сравнивает две строки и возвращает 1, если первая строка больше второй; 0 - если строки равны; -1 (минус один) - если первая строка меньше второй	<a href="#">Строковые функции</a>
<a href="#">StringConcatenate</a>	Формирует строку из переданных параметров	<a href="#">Строковые функции</a>
<a href="#">StringFill</a>	Заполняет указанную строку по месту указанными символами	<a href="#">Строковые функции</a>
<a href="#">StringFind</a>	Поиск подстроки в строке	<a href="#">Строковые функции</a>
<a href="#">StringFormat</a>	Преобразует число в строку в соответствие с заданным форматом	<a href="#">Преобразование данных</a>
<a href="#">StringGetCharacter</a>	Возвращает значение символа, расположенного в указанной позиции строки	<a href="#">Строковые функции</a>
<a href="#">StringInit</a>	Инициализирует строку указанными символами и обеспечивает указанный размер строки	<a href="#">Строковые функции</a>
<a href="#">StringLen</a>	Возвращает число символов в строке	<a href="#">Строковые функции</a>
<a href="#">StringReplace</a>	Заменяет в строке все найденные подстроки на заданную последовательность символов.	<a href="#">Строковые функции</a>
<a href="#">StringSetCharacter</a>	Возвращает копию строки с измененным значением	<a href="#">Строковые функции</a>

	символа в указанной позиции	
<a href="#">StringSplit</a>	Получает из указанной строки подстроки по заданному разделителю и возвращает количество полученных подстрок	<a href="#">Строковые функции</a>
<a href="#">StringSubstr</a>	Извлекает подстроку из текстовой строки, начинающейся с указанной позиции	<a href="#">Строковые функции</a>
<a href="#">StringToArray</a>	Посимвольно копирует преобразованную из Unicode в ANSI строку в указанное место массива типа uchar	<a href="#">Преобразование данных</a>
<a href="#">StringToColor</a>	Преобразует строку типа "R,G,B" или строку, содержащую наименование цвета, в значение типа color	<a href="#">Преобразование данных</a>
<a href="#">StringtoDouble</a>	Преобразование строки, содержащей символьное представление числа, в число типа double	<a href="#">Преобразование данных</a>
<a href="#">StringToInteger</a>	Преобразование строки, содержащей символьное представление числа, в число типа int	<a href="#">Преобразование данных</a>
<a href="#">StringToLower</a>	Преобразует все символы указанной строки в строчные (маленькие) по месту	<a href="#">Строковые функции</a>
<a href="#">StringToShortArray</a>	Посимвольно копирует строку в указанное место массива типа ushort	<a href="#">Преобразование данных</a>
<a href="#">StringToTime</a>	Преобразование строки, содержащей время и/или дату в формате "yyyy.mm.dd [hh:mi]", в число типа datetime	<a href="#">Преобразование данных</a>
<a href="#">StringToUpper</a>	Преобразует все символы указанной строки в прописные (большие) по месту	<a href="#">Строковые функции</a>
<a href="#">StringTrimLeft</a>	Удаляет символы перевода каретки, пробелы и символы	<a href="#">Строковые функции</a>

	табуляции с начала строки	
<a href="#">StringTrimRight</a>	Удаляет символы перевода каретки, пробелы и символы табуляции в конце строки	<a href="#">Строковые функции</a>
<a href="#">StructToTime</a>	Производит конвертацию из переменной типа структуры <code>MqlDateTime</code> в значение типа <code>datetime</code>	<a href="#">Дата и время</a>
<a href="#">Symbol</a>	Возвращает имя символа текущего графика.	<a href="#">Проверка состояния</a>
<a href="#">SymbolInfoDouble</a>	Возвращает значение типа <code>double</code> указанного символа для соответствующего свойства	<a href="#">Получение рыночной информации</a>
<a href="#">SymbolInfoInteger</a>	Возвращает значение целочисленного типа ( <code>long</code> , <code>datetime</code> , <code>int</code> или <code>bool</code> ) указанного символа для соответствующего свойства	<a href="#">Получение рыночной информации</a>
<a href="#">SymbolInfoMarginRate</a>	Возвращает коэффициенты взимания маржи в зависимости от типа и направления ордера	<a href="#">Получение рыночной информации</a>
<a href="#">SymbolInfoSessionQuote</a>	Позволяет получить время начала и время окончания указанной котировочной сессии для указанных символа и дня недели.	<a href="#">Получение рыночной информации</a>
<a href="#">SymbolInfoSessionTrade</a>	Позволяет получить время начала и время окончания указанной торговой сессии для указанных символа и дня недели.	<a href="#">Получение рыночной информации</a>
<a href="#">SymbolInfoString</a>	Возвращает значение типа <code>string</code> указанного символа для соответствующего свойства	<a href="#">Получение рыночной информации</a>
<a href="#">SymbolInfoTick</a>	Возвращает текущие цены для указанного символа в переменной типа <code>MqlTick</code>	<a href="#">Получение рыночной информации</a>
<a href="#">SymbolsSynchronized</a>	Проверяет <a href="#">синхронизированность</a> данных по указанному	<a href="#">Получение рыночной информации</a>

	символу в терминале с данными на торговом сервере	
<a href="#">SymbolName</a>	Возвращает наименование указанного символа	<a href="#">Получение рыночной информации</a>
<a href="#">SymbolSelect</a>	Выбирает символ в окне MarketWatch или убирает символ из окна	<a href="#">Получение рыночной информации</a>
<a href="#">SymbolsTotal</a>	Возвращает количество доступных (выбранных в MarketWatch или всех) символов	<a href="#">Получение рыночной информации</a>
<a href="#">tan</a>	Возвращает тангенс числа	<a href="#">Математические функции</a>
<a href="#">TerminalClose</a>	Посыпает терминалу команду на завершение работы	<a href="#">Общие функции</a>
<a href="#">TerminalInfoDouble</a>	Возвращает значение типа double соответствующего свойства окружения mql5-программы	<a href="#">Проверка состояния</a>
<a href="#">TerminalInfoInteger</a>	Возвращает значение целого типа соответствующего свойства окружения mql5-программы	<a href="#">Проверка состояния</a>
<a href="#">TerminalInfoString</a>	Возвращает значение типа string соответствующего свойства окружения mql5-программы	<a href="#">Проверка состояния</a>
<a href="#">TesterStatistics</a>	Возвращает значение указанного статистического показателя, рассчитанного по результатам тестирования	<a href="#">Общие функции</a>
<a href="#">TextGetSize</a>	Возвращает ширину и высоту строки при текущих <a href="#">настройках шрифта</a>	<a href="#">Графические объекты</a>
<a href="#">TextOut</a>	Выводит текст в пользовательский массив (буфер), предназначенный для создания графического <a href="#">ресурса</a>	<a href="#">Графические объекты</a>
<a href="#">TextSetFont</a>	Устанавливает шрифт для вывода текста методами рисования (по умолчанию используется шрифт Arial 20)	<a href="#">Графические объекты</a>

<a href="#">TimeCurrent</a>	Возвращает последнее известное время сервера (время прихода последней котировки) в формате datetime	<a href="#">Дата и время</a>
<a href="#">TimeDaylightSavings</a>	Возвращает признак перехода на летнее /зимнее время	<a href="#">Дата и время</a>
<a href="#">TimeGMT</a>	Возвращает время GMT формате datetime с учетом перехода на зимнее или летнее время по локальному времени компьютера, на котором запущен клиентский терминал	<a href="#">Дата и время</a>
<a href="#">TimeGMTOffset</a>	Возвращает текущую разницу между временем GMT и локальным временем компьютера в секундах с учетом перехода на зимнее или летнее время	<a href="#">Дата и время</a>
<a href="#">TimeLocal</a>	Возвращает локальное компьютерное время в формате datetime	<a href="#">Дата и время</a>
<a href="#">TimeToString</a>	Преобразование значения, содержащего время в секундах, прошедшее с 01.01.1970, в строку формата "уууу.мм.дд hh:mi"	<a href="#">Преобразование данных</a>
<a href="#">TimeToStruct</a>	Производит конвертацию из значения типа datetime в переменную типа структуры MqlDateTime	<a href="#">Дата и время</a>
<a href="#">TimeTradeServer</a>	Возвращает расчетное текущее время торгового сервера	<a href="#">Дата и время</a>
<a href="#">UninitializeReason</a>	Возвращает код причины деинициализации	<a href="#">Проверка состояния</a>
<a href="#">WebRequest</a>	Отправляет HTTP-запрос на указанный сервер	<a href="#">Общие функции</a>
<a href="#">ZeroMemory</a>	Обнуляет переменную, переданную по ссылке. Тип переменной может быть любым, исключение	<a href="#">Общие функции</a>

	составляют только классы и структуры, имеющие конструкторы	
--	--	--

## Список констант языка MQL5

Все константы MQL5 в алфавитном порядке.

Константа	Описание	Использование
<code>_DATE_</code>	Дата компиляции файла без времени (часы, минуты и секунды равны 0)	<a href="#">Print</a>
<code>_DATETIME_</code>	Дата и время компиляции файла	<a href="#">Print</a>
<code>_FILE_</code>	Имя текущего компилируемого файла	<a href="#">Print</a>
<code>_FUNCSIG_</code>	Сигнатура функции, в теле которой расположен макрос. Вывод в лог полного описания функции с типами параметров может пригодиться при идентификации <a href="#">перегруженных функций</a>	<a href="#">Print</a>
<code>_FUNCTION_</code>	Имя функции, в теле которой расположен макрос	<a href="#">Print</a>
<code>_LINE_</code>	Номер строки в исходном коде, на которой расположен данный макрос	<a href="#">Print</a>
<code>_MQLBUILD_, _MQL5BUILD_</code>	Номер билда компилятора	<a href="#">Print</a>
<code>_PATH_</code>	Абсолютный путь к текущему компилируемому файлу	<a href="#">Print</a>
<code>ACCOUNT_ASSETS</code>	Текущий размер активов на счёте	<a href="#">AccountInfoDouble</a>
<code>ACCOUNT_BALANCE</code>	Баланс счета в валюте депозита	<a href="#">AccountInfoDouble</a>
<code>ACCOUNT_COMMISSION_BLO KED</code>	Текущая сумма заблокированных комиссий по счёту	<a href="#">AccountInfoDouble</a>
<code>ACCOUNT_COMPANY</code>	Имя компании, обслуживающей счет	<a href="#">AccountInfoString</a>
<code>ACCOUNT_CREDIT</code>	Размер предоставленного кредита в валюте депозита	<a href="#">AccountInfoDouble</a>
<code>ACCOUNT_CURRENCY</code>	Валюта депозита	<a href="#">AccountInfoString</a>

ACCOUNT_EQUITY	Значение собственных средств на счете в валюте депозита	<a href="#">AccountInfoDouble</a>
ACCOUNT_LEVERAGE	Размер предоставленного плеча	<a href="#">AccountInfoInteger</a>
ACCOUNT_LIABILITIES	Текущий размер обязательств на счёте	<a href="#">AccountInfoDouble</a>
ACCOUNT_LIMIT_ORDERS	Максимально допустимое количество действующих отложенных ордеров	<a href="#">AccountInfoInteger</a>
ACCOUNT_LOGIN	Номер счета	<a href="#">AccountInfoInteger</a>
ACCOUNT_MARGIN	Размер зарезервированных залоговых средств на счете в валюте депозита	<a href="#">AccountInfoDouble</a>
ACCOUNT_MARGIN_FREE	Размер свободных средств на счете в валюте депозита, доступных для открытия позиции	<a href="#">AccountInfoDouble</a>
ACCOUNT_MARGIN_INITIAL	Размер средств, зарезервированных на счёте, для обеспечения гарантийной суммы по всем отложенным ордерам	<a href="#">AccountInfoDouble</a>
ACCOUNT_MARGIN_LEVEL	Уровень залоговых средств на счете в процентах	<a href="#">AccountInfoDouble</a>
ACCOUNT_MARGIN_MAINTENANCE	Размер средств, зарезервированных на счёте, для обеспечения минимальной суммы по всем открытым позициям	<a href="#">AccountInfoDouble</a>
ACCOUNT_MARGIN_SO_CALL	Уровень залоговых средств, при котором требуется пополнение счета (Margin Call). В зависимости от установленного ACCOUNT_MARGIN_SO_MODE выражается в процентах либо в валюте депозита	<a href="#">AccountInfoDouble</a>
ACCOUNT_MARGIN_SO_MODE	Режим задания минимально допустимого уровня залоговых средств	<a href="#">AccountInfoInteger</a>
ACCOUNT_MARGIN_SO_SO	Уровень залоговых средств, при достижении которого	<a href="#">AccountInfoDouble</a>

	происходит принудительное закрытие самой убыточной позиции (Stop Out). В зависимости от установленного ACCOUNT_MARGIN_SO_MODE выражается в процентах либо в валюте депозита	
ACCOUNT_NAME	Имя клиента	<a href="#">AccountInfoString</a>
ACCOUNT_PROFIT	Размер текущей прибыли на счете в валюте депозита	<a href="#">AccountInfoDouble</a>
ACCOUNT_SERVER	Имя торгового сервера	<a href="#">AccountInfoString</a>
ACCOUNT_STOPOUT_MODE_MONEY	Уровень задается в деньгах	<a href="#">AccountInfoInteger</a>
ACCOUNT_STOPOUT_MODE_PERCENT	Уровень задается в процентах	<a href="#">AccountInfoInteger</a>
ACCOUNT_TRADE_ALLOWED	<a href="#">Разрешенность торговли</a> для текущего счета	<a href="#">AccountInfoInteger</a>
ACCOUNT_TRADE_EXPERT	<a href="#">Разрешенность торговли</a> для эксперта	<a href="#">AccountInfoInteger</a>
ACCOUNT_TRADE_MODE	Тип торгового счета	<a href="#">AccountInfoInteger</a>
ACCOUNT_TRADE_MODE_CONTEST	Конкурсный торговый счет	<a href="#">AccountInfoInteger</a>
ACCOUNT_TRADE_MODE_DEMO	Демонстрационный торговый счет	<a href="#">AccountInfoInteger</a>
ACCOUNT_TRADE_MODE_REAL	Реальный торговый счет	<a href="#">AccountInfoInteger</a>
ALIGN_CENTER	Выравнивание по центру (только для объекта "Поле ввода")	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a> , <a href="#">ChartScreenShot</a>
ALIGN_LEFT	Выравнивание по левой границе	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a> , <a href="#">ChartScreenShot</a>
ALIGN_RIGHT	Выравнивание по правой границе	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a> , <a href="#">ChartScreenShot</a>
ANCHOR_CENTER	Точка привязки строго по центру объекта	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
ANCHOR_LEFT	Точка привязки слева по центру	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>

ANCHOR_LEFT_LOWER	Точка привязки в левом нижнем углу	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
ANCHOR_LEFT_UPPER	Точка привязки в левом верхнем углу	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
ANCHOR_LOWER	Точка привязки снизу по центру	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
ANCHOR_RIGHT	Точка привязки справа по центру	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
ANCHOR_RIGHT_LOWER	Точка привязки в правом нижнем углу	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
ANCHOR_RIGHT_UPPER	Точка привязки в правом верхнем углу	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
ANCHOR_UPPER	Точка привязки сверху по центру	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
BASE_LINE	Основная линия	Линии индикаторов
BOOK_TYPE_BUY	Заявка на покупку	<a href="#">MqlBookInfo</a>
BOOK_TYPE_BUY_MARKET	Заявка на покупку по рыночной цене	<a href="#">MqlBookInfo</a>
BOOK_TYPE_SELL	Заявка на продажу	<a href="#">MqlBookInfo</a>
BOOK_TYPE_SELL_MARKET	Заявка на продажу по рыночной цене	<a href="#">MqlBookInfo</a>
BORDER_FLAT	Плоский вид	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
BORDER_RAISED	Выпуклый вид	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
BORDER_SUNKEN	Вогнутый вид	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
CHAR_MAX	Максимальное значение, которое может быть представлено типом char	Константы числовых типов
CHAR_MIN	Минимальное значение, которое может быть представлено типом char	Константы числовых типов
<a href="#">CHART_AUTOSCROLL</a>	Режим автоматического перехода к правому краю графика	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
CHART_BARS	Отображение в виде баров	<a href="#">ChartSetInteger</a>
CHART_BEGIN	Начало графика (самые старые цены)	<a href="#">ChartNavigate</a>

<a href="#">CHART_BRING_TO_TOP</a>	Показ графика поверх всех других	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_CANDLES</a>	Отображение в виде японских свечей	<a href="#">ChartSetInteger</a>
<a href="#">CHART_COLOR_ASK</a>	Цвет линии Ask-цены	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_BACKGROUND</a>	Цвет фона графика	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_BID</a>	Цвет линии Bid-цены	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_CANDLE_BEAR</a>	Цвет тела медвежьей свечи	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_CANDLE_BULL</a>	Цвет тела бычьей свечи	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_CHART_DOWN</a>	Цвет бара вниз, тени и окантовки тела медвежьей свечи	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_CHART_LINE</a>	Цвет линии графика и японских свечей "Доджи"	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_CHART_UP</a>	Цвет бара вверх, тени и окантовки тела бычьей свечи	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_FOREGROUND</a>	Цвет осей, шкалы и строки OHLC	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_GRID</a>	Цвет сетки	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_LAST</a>	Цвет линии цены последней совершенной сделки (Last)	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_STOP_LEVEL</a>	Цвет уровней стоп-ордеров (Stop Loss и Take Profit)	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COLOR_VOLUME</a>	Цвет объемов и уровней открытия позиций	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_COMMENT</a>	Текст комментария на графике	<a href="#">ChartSetString</a> , <a href="#">ChartGetString</a>
<a href="#">CHART_CURRENT_POS</a>	Текущая позиция	<a href="#">ChartNavigate</a>
<a href="#">CHART_DRAG_TRADE_LEVELS</a>	Разрешение на перетаскивание торговых уровней на графике с помощью мышки. По умолчанию режим	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>

	перетаскивания включен (значение true)	
<a href="#">CHART_EVENT_MOUSE_MOVE</a>	Отправка всем mql5-программам на графике сообщений о событиях перемещения и нажатия кнопок мыши ( <a href="#">CHARTEVENT_MOUSE_MOVE</a> )	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_EVENT_OBJECT_CREATE</a>	Отправка всем mql5-программам на графике сообщений о событии создания графического объекта ( <a href="#">CHARTEVENT_OBJECT_CREATE</a> )	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_EVENT_OBJECT_DELETE</a>	Отправка всем mql5-программам на графике сообщений о событии удаления графического объекта ( <a href="#">CHARTEVENT_OBJECT_DELETE</a> )	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_FIRST_VISIBLE_BAR</a>	Номер первого видимого бара на графике. Индексация баров соответствует <a href="#">таймсерии</a> .	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_FIXED_MAX</a>	Фиксированный максимум графика	<a href="#">ChartSetDouble</a> , <a href="#">ChartGetDouble</a>
<a href="#">CHART_FIXED_MIN</a>	Фиксированный минимум графика	<a href="#">ChartSetDouble</a> , <a href="#">ChartGetDouble</a>
<a href="#">CHART_FIXED_POSITION</a>	Положение фиксированной позиции графика от левого края в процентах. Фиксированная позиция графика обозначена маленьким серым треугольником на горизонтальной оси времени и показывается только в том случае, если отключена автоматическая прокрутка к правому краю при поступлении нового тика (смотри свойство <a href="#">CHART_AUTOSCROLL</a> ). Бар, который находится на	<a href="#">ChartSetDouble</a> , <a href="#">ChartGetDouble</a>

	фиксированной позиции, остаётся на том же месте при увеличении и уменьшении масштаба.	
<a href="#">CHART_FOREGROUND</a>	Ценовой график на переднем плане	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_HEIGHT_IN_PIXELS</a>	Высота графика в пикселях	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_IS_OBJECT</a>	Признак для идентификации объекта "График" ( <a href="#">OBJ_CHART</a> ) - для графического объекта возвращает true. Для настоящего графика значение равно false	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_LINE</a>	Отображение в виде линии, проведенной по ценам Close	<a href="#">ChartSetInteger</a>
<a href="#">CHART_MODE</a>	Тип графика (свечи, бары или линия)	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_MOUSE_SCROLL</a>	Прокрутка графика левой кнопкой мышки по горизонтали. Прокрутка по вертикали также будет доступна, если установлено в true значение любого из трех свойств: CHART_SCALEFIX, CHART_SCALEFIX_11 или CHART_SCALE_PT_PER_BAR	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_POINTS_PER_BAR</a>	Значение масштаба в пунктах на бар	<a href="#">ChartSetDouble</a> , <a href="#">ChartGetDouble</a>
<a href="#">CHART_PRICE_MAX</a>	Максимум графика	<a href="#">ChartSetDouble</a> , <a href="#">ChartGetDouble</a>
<a href="#">CHART_PRICE_MIN</a>	Минимум графика	<a href="#">ChartSetDouble</a> , <a href="#">ChartGetDouble</a>
<a href="#">CHART_SCALE</a>	Масштаб	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SCALE_PT_PER_BAR</a>	Режим указания масштаба в пунктах на бар	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SCALEFIX</a>	Режим фиксированного масштаба	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SCALEFIX_11</a>	Режим масштаба 1:1	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>

<a href="#">CHART_SHIFT</a>	Режим отступа ценового графика от правого края	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHIFT_SIZE</a>	Размер отступа нулевого бара от правого края в процентах	<a href="#">ChartSetDouble</a> , <a href="#">ChartGetDouble</a>
<a href="#">CHART_SHOW_ASK_LINE</a>	Отображение значения Ask горизонтальной линией на графике	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_BID_LINE</a>	Отображение значения Bid горизонтальной линией на графике	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_DATE_SCALE</a>	Отображение на графике шкалы времени	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_GRID</a>	Отображение сетки на графике	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_LAST_LINE</a>	Отображение значения Last горизонтальной линией на графике	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_OBJECT_DESCR</a>	Всплывающие описания графических объектов	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_OHLC</a>	Отображение в левом верхнем углу значений OHLC	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_ONE_CLICK</a>	Отображение на графике панели быстрой торговли (опция " <a href="#">Торговля одним кликом</a> ")	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_PERIOD_SEP</a>	Отображение вертикальных разделителей между соседними периодами	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_PRICE_SCALE</a>	Отображение на графике ценовой шкалы	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_TRADE_LEVELS</a>	Отображение на графике торговых уровней (уровни открытых позиций, Stop Loss, Take Profit и отложенных ордеров)	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_SHOW_VOLUMES</a>	Отображение объемов на графике	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_VISIBLE_BARS</a>	Количество баров на графике, доступных для отображения	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<a href="#">CHART_VOLUME_HIDE</a>	Объемы не показаны	<a href="#">ChartSetInteger</a>

<code>CHART_VOLUME_REAL</code>	Торговые объемы	<a href="#">ChartSetInteger</a>
<code>CHART_VOLUME_TICK</code>	Тиковые объемы	<a href="#">ChartSetInteger</a>
<code>CHART_WIDTH_IN_BARS</code>	Ширина графика в барах	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<code>CHART_WIDTH_IN_PIXELS</code>	Ширина графика в пикселях	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<code>CHART_WINDOW_HANDLE</code>	Хэндл графика (HWND)	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<code>CHART_WINDOW_IS_VISIBLE</code>	Видимость подокон	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<code>CHART_WINDOW_YDISTANCE</code>	<p>Дистанция в пикселях по вертикальной оси Y между верхней рамкой подокна индикатора и верхней рамкой главного окна графика. При наступлении событий мыши координаты курсора передаются в координатах главного окна графика, в то время как координаты графических объектов в подокне индикатора задаются относительно верхнего левого угла подокна.</p> <p>Значение требуется для перевода абсолютных координат главного графика в локальные координаты подокна для корректной работы с графическими объектами, у которых координаты задаются относительно верхнего левого угла рамки подокна.</p>	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<code>CHART_WINDOWS_TOTAL</code>	Общее количество окон графика, включая подокна индикаторов	<a href="#">ChartSetInteger</a> , <a href="#">ChartGetInteger</a>
<code>CHARTEVENT_CHART_CHANGE</code>	Изменение размеров графика или изменение свойств графика через диалог свойств	<a href="#">OnChartEvent</a>
<code>CHARTEVENT_CLICK</code>	Нажатие мышки на графике	<a href="#">OnChartEvent</a>

CHARTEVENT_CUSTOM	Начальный номер события из диапазона пользовательских событий	<a href="#">OnChartEvent</a>
CHARTEVENT_CUSTOM_LAST	Конечный номер события из диапазона пользовательских событий	<a href="#">OnChartEvent</a>
CHARTEVENT_KEYDOWN	Нажатие клавиатуры	<a href="#">OnChartEvent</a>
CHARTEVENT_MOUSE_MOVE	Перемещение мыши и нажатие кнопок мыши (если для графика установлено свойство <a href="#">CHART_EVENT_MOUSE_MOVE=true</a> )	<a href="#">OnChartEvent</a>
CHARTEVENT_OBJECT_CHANGE	Изменение свойств <a href="#">графического объекта</a> через диалог свойств	<a href="#">OnChartEvent</a>
CHARTEVENT_OBJECT_CLICK	Нажатие мышки на <a href="#">графическом объекте</a>	<a href="#">OnChartEvent</a>
CHARTEVENT_OBJECT_CREATE	Создание <a href="#">графического объекта</a> (если для графика установлено свойство <a href="#">CHART_EVENT_OBJECT_CREATE=true</a> )	<a href="#">OnChartEvent</a>
CHARTEVENT_OBJECT_DELETE	Удаление <a href="#">графического объекта</a> (если для графика установлено свойство <a href="#">CHART_EVENT_OBJECT_DELETE=true</a> )	<a href="#">OnChartEvent</a>
CHARTEVENT_OBJECT_DRAG	Перетаскивание <a href="#">графического объекта</a>	<a href="#">OnChartEvent</a>
CHARTEVENT_OBJECT_ENDEDIT	Окончание редактирования текста в <a href="#">графическом объекте Edit</a>	<a href="#">OnChartEvent</a>
CHARTS_MAX	Максимально возможное количество одновременно открытых графиков в терминале	<a href="#">Прочие константы</a>
CHIKOUSPAN_LINE	Линия Chikou Span	<a href="#">Линии индикаторов</a>
clrAliceBlue	Alice Blue	<a href="#">Набор Web-цветов</a>
clrAntiqueWhite	Antique White	<a href="#">Набор Web-цветов</a>
clrAqua	Aqua	<a href="#">Набор Web-цветов</a>

clrAquamarine	Aquamarine	<a href="#">Набор Web-цветов</a>
clrBeige	Beige	<a href="#">Набор Web-цветов</a>
clrBisque	Bisque	<a href="#">Набор Web-цветов</a>
clrBlack	Black	<a href="#">Набор Web-цветов</a>
clrBlanchedAlmond	Blanched Almond	<a href="#">Набор Web-цветов</a>
clrBlue	Blue	<a href="#">Набор Web-цветов</a>
clrBlueViolet	Blue Violet	<a href="#">Набор Web-цветов</a>
clrBrown	Brown	<a href="#">Набор Web-цветов</a>
clrBurlyWood	Burly Wood	<a href="#">Набор Web-цветов</a>
clrCadetBlue	Cadet Blue	<a href="#">Набор Web-цветов</a>
clrChartreuse	Chartreuse	<a href="#">Набор Web-цветов</a>
clrChocolate	Chocolate	<a href="#">Набор Web-цветов</a>
clrCoral	Coral	<a href="#">Набор Web-цветов</a>
clrCornflowerBlue	Cornflower Blue	<a href="#">Набор Web-цветов</a>
clrCornsilk	Cornsilk	<a href="#">Набор Web-цветов</a>
clrCrimson	Crimson	<a href="#">Набор Web-цветов</a>
clrDarkBlue	Dark Blue	<a href="#">Набор Web-цветов</a>
clrDarkGoldenrod	Dark Goldenrod	<a href="#">Набор Web-цветов</a>
clrDarkGray	Dark Gray	<a href="#">Набор Web-цветов</a>
clrDarkGreen	Dark Green	<a href="#">Набор Web-цветов</a>
clrDarkKhaki	Dark Khaki	<a href="#">Набор Web-цветов</a>
clrDarkOliveGreen	Dark Olive Green	<a href="#">Набор Web-цветов</a>
clrDarkOrange	Dark Orange	<a href="#">Набор Web-цветов</a>
clrDarkOrchid	Dark Orchid	<a href="#">Набор Web-цветов</a>
clrDarkSalmon	Dark Salmon	<a href="#">Набор Web-цветов</a>
clrDarkSeaGreen	Dark Sea Green	<a href="#">Набор Web-цветов</a>
clrDarkSlateBlue	Dark Slate Blue	<a href="#">Набор Web-цветов</a>
clrDarkSlateGray	Dark Slate Gray	<a href="#">Набор Web-цветов</a>
clrDarkTurquoise	Dark Turquoise	<a href="#">Набор Web-цветов</a>
clrDarkViolet	Dark Violet	<a href="#">Набор Web-цветов</a>
clrDeepPink	Deep Pink	<a href="#">Набор Web-цветов</a>

clrDeepSkyBlue	Deep Sky Blue	<a href="#">Набор Web-цветов</a>
clrDimGray	Dim Gray	<a href="#">Набор Web-цветов</a>
clrDodgerBlue	Dodger Blue	<a href="#">Набор Web-цветов</a>
clrFireBrick	Fire Brick	<a href="#">Набор Web-цветов</a>
clrForestGreen	Forest Green	<a href="#">Набор Web-цветов</a>
clrGainsboro	Gainsboro	<a href="#">Набор Web-цветов</a>
clrGold	Gold	<a href="#">Набор Web-цветов</a>
clrGoldenrod	Goldenrod	<a href="#">Набор Web-цветов</a>
clrGray	Gray	<a href="#">Набор Web-цветов</a>
clrGreen	Green	<a href="#">Набор Web-цветов</a>
clrGreenYellow	Green Yellow	<a href="#">Набор Web-цветов</a>
clrHoneydew	Honeydew	<a href="#">Набор Web-цветов</a>
clrHotPink	Hot Pink	<a href="#">Набор Web-цветов</a>
clrIndianRed	Indian Red	<a href="#">Набор Web-цветов</a>
clrIndigo	Indigo	<a href="#">Набор Web-цветов</a>
clrIvory	Ivory	<a href="#">Набор Web-цветов</a>
clrKhaki	Khaki	<a href="#">Набор Web-цветов</a>
clrLavender	Lavender	<a href="#">Набор Web-цветов</a>
clrLavenderBlush	Lavender Blush	<a href="#">Набор Web-цветов</a>
clrLawnGreen	Lawn Green	<a href="#">Набор Web-цветов</a>
clrLemonChiffon	Lemon Chiffon	<a href="#">Набор Web-цветов</a>
clrLightBlue	Light Blue	<a href="#">Набор Web-цветов</a>
clrLightCoral	Light Coral	<a href="#">Набор Web-цветов</a>
clrLightCyan	Light Cyan	<a href="#">Набор Web-цветов</a>
clrLightGoldenrod	Light Goldenrod	<a href="#">Набор Web-цветов</a>
clrLightGray	Light Gray	<a href="#">Набор Web-цветов</a>
clrLightGreen	Light Green	<a href="#">Набор Web-цветов</a>
clrLightPink	Light Pink	<a href="#">Набор Web-цветов</a>
clrLightSalmon	Light Salmon	<a href="#">Набор Web-цветов</a>
clrLightSeaGreen	Light Sea Green	<a href="#">Набор Web-цветов</a>
clrLightSkyBlue	Light Sky Blue	<a href="#">Набор Web-цветов</a>

clrLightSlateGray	Light Slate Gray	<a href="#">Набор Web-цветов</a>
clrLightSteelBlue	Light Steel Blue	<a href="#">Набор Web-цветов</a>
clrLightYellow	Light Yellow	<a href="#">Набор Web-цветов</a>
clrLime	Lime	<a href="#">Набор Web-цветов</a>
clrLimeGreen	Lime Green	<a href="#">Набор Web-цветов</a>
clrLinen	Linen	<a href="#">Набор Web-цветов</a>
clrMagenta	Magenta	<a href="#">Набор Web-цветов</a>
clrMaroon	Maroon	<a href="#">Набор Web-цветов</a>
clrMediumAquamarine	Medium Aquamarine	<a href="#">Набор Web-цветов</a>
clrMediumBlue	Medium Blue	<a href="#">Набор Web-цветов</a>
clrMediumOrchid	Medium Orchid	<a href="#">Набор Web-цветов</a>
clrMediumPurple	Medium Purple	<a href="#">Набор Web-цветов</a>
clrMediumSeaGreen	Medium Sea Green	<a href="#">Набор Web-цветов</a>
clrMediumSlateBlue	Medium Slate Blue	<a href="#">Набор Web-цветов</a>
clrMediumSpringGreen	Medium Spring Green	<a href="#">Набор Web-цветов</a>
clrMediumTurquoise	Medium Turquoise	<a href="#">Набор Web-цветов</a>
clrMediumVioletRed	Medium Violet Red	<a href="#">Набор Web-цветов</a>
clrMidnightBlue	Midnight Blue	<a href="#">Набор Web-цветов</a>
clrMintCream	Mint Cream	<a href="#">Набор Web-цветов</a>
clrMistyRose	Misty Rose	<a href="#">Набор Web-цветов</a>
clrMoccasin	Moccasin	<a href="#">Набор Web-цветов</a>
clrNavajoWhite	Navajo White	<a href="#">Набор Web-цветов</a>
clrNavy	Navy	<a href="#">Набор Web-цветов</a>
clrNONE	Отсутствие цвета	<a href="#">Прочие константы</a>
clrOldLace	Old Lace	<a href="#">Набор Web-цветов</a>
clrOlive	Olive	<a href="#">Набор Web-цветов</a>
clrOliveDrab	Olive Drab	<a href="#">Набор Web-цветов</a>
clrOrange	Orange	<a href="#">Набор Web-цветов</a>
clrOrangeRed	Orange Red	<a href="#">Набор Web-цветов</a>
clrOrchid	Orchid	<a href="#">Набор Web-цветов</a>
clrPaleGoldenrod	Pale Goldenrod	<a href="#">Набор Web-цветов</a>

clrPaleGreen	Pale Green	<a href="#">Набор Web-цветов</a>
clrPaleTurquoise	Pale Turquoise	<a href="#">Набор Web-цветов</a>
clrPaleVioletRed	Pale Violet Red	<a href="#">Набор Web-цветов</a>
clrPapayaWhip	Papaya Whip	<a href="#">Набор Web-цветов</a>
clrPeachPuff	Peach Puff	<a href="#">Набор Web-цветов</a>
clrPeru	Peru	<a href="#">Набор Web-цветов</a>
clrPink	Pink	<a href="#">Набор Web-цветов</a>
clrPlum	Plum	<a href="#">Набор Web-цветов</a>
clrPowderBlue	Powder Blue	<a href="#">Набор Web-цветов</a>
clrPurple	Purple	<a href="#">Набор Web-цветов</a>
clrRed	Red	<a href="#">Набор Web-цветов</a>
clrRosyBrown	Rosy Brown	<a href="#">Набор Web-цветов</a>
clrRoyalBlue	Royal Blue	<a href="#">Набор Web-цветов</a>
clrSaddleBrown	Saddle Brown	<a href="#">Набор Web-цветов</a>
clrSalmon	Salmon	<a href="#">Набор Web-цветов</a>
clrSandyBrown	Sandy Brown	<a href="#">Набор Web-цветов</a>
clrSeaGreen	Sea Green	<a href="#">Набор Web-цветов</a>
clrSeashell	Seashell	<a href="#">Набор Web-цветов</a>
clrSienna	Sienna	<a href="#">Набор Web-цветов</a>
clrSilver	Silver	<a href="#">Набор Web-цветов</a>
clrSkyBlue	Sky Blue	<a href="#">Набор Web-цветов</a>
clrSlateBlue	Slate Blue	<a href="#">Набор Web-цветов</a>
clrSlateGray	Slate Gray	<a href="#">Набор Web-цветов</a>
clrSnow	Snow	<a href="#">Набор Web-цветов</a>
clrSpringGreen	Spring Green	<a href="#">Набор Web-цветов</a>
clrSteelBlue	Steel Blue	<a href="#">Набор Web-цветов</a>
clrTan	Tan	<a href="#">Набор Web-цветов</a>
clrTeal	Teal	<a href="#">Набор Web-цветов</a>
clrThistle	Thistle	<a href="#">Набор Web-цветов</a>
clrTomato	Tomato	<a href="#">Набор Web-цветов</a>
clrTurquoise	Turquoise	<a href="#">Набор Web-цветов</a>

clrViolet	Violet	<a href="#">Набор Web-цветов</a>
clrWheat	Wheat	<a href="#">Набор Web-цветов</a>
clrWhite	White	<a href="#">Набор Web-цветов</a>
clrWhiteSmoke	White Smoke	<a href="#">Набор Web-цветов</a>
clrYellow	Yellow	<a href="#">Набор Web-цветов</a>
clrYellowGreen	Yellow Green	<a href="#">Набор Web-цветов</a>
CORNER_LEFT_LOWER	Центр координат в левом нижнем углу графика	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
CORNER_LEFT_UPPER	Центр координат в левом верхнем углу графика	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
CORNER_RIGHT_LOWER	Центр координат в правом нижнем углу графика	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
CORNER_RIGHT_UPPER	Центр координат в правом верхнем углу графика	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
CP_ACP	Текущая кодовая страница ANSI кодировка в операционной системе Windows	<a href="#">CharArrayToString</a> , <a href="#">StringToCharArray</a> , <a href="#">FileOpen</a>
CP_MACCP	Текущая кодовая страница Macintosh. Примечание: Это значение преимущественно используется в ранее созданных программных кодах и теперь в нем нет необходимости, так как современные компьютеры Macintosh используют Unicode кодировку.	<a href="#">CharArrayToString</a> , <a href="#">StringToCharArray</a> , <a href="#">FileOpen</a>
CP_OEMCP	Текущая кодовая страница OEM.	<a href="#">CharArrayToString</a> , <a href="#">StringToCharArray</a> , <a href="#">FileOpen</a>
CP_SYMBOL	Кодовая страница Symbol	<a href="#">CharArrayToString</a> , <a href="#">StringToCharArray</a> , <a href="#">FileOpen</a>
CP_THREAD_ACP	Кодировка Windows ANSI для текущего потока выполнения.	<a href="#">CharArrayToString</a> , <a href="#">StringToCharArray</a> , <a href="#">FileOpen</a>
CP_UTF7	Кодовая страница UTF-7.	<a href="#">CharArrayToString</a> , <a href="#">StringToCharArray</a> , <a href="#">FileOpen</a>
CP_UTF8	Кодовая страница UTF-8.	<a href="#">CharArrayToString</a> , <a href="#">StringToCharArray</a> , <a href="#">FileOpen</a>

CRYPT_AES128	Шифрование AES с ключом 128 бит (16 байт)	<a href="#">CryptEncode</a> , <a href="#">CryptDecode</a>
CRYPT_AES256	Шифрование AES с ключом 256 бит (32 байта)	<a href="#">CryptEncode</a> , <a href="#">CryptDecode</a>
CRYPT_ARCH_ZIP	ZIP архивирование	<a href="#">CryptEncode</a> , <a href="#">CryptDecode</a>
CRYPT_BASE64	Шифрование BASE64 (перекодировка)	<a href="#">CryptEncode</a> , <a href="#">CryptDecode</a>
CRYPT_DES	Шифрование DES с ключом 56 бит (7 байт)	<a href="#">CryptEncode</a> , <a href="#">CryptDecode</a>
CRYPT_HASH_MD5	Расчёт HASH MD5	<a href="#">CryptEncode</a> , <a href="#">CryptDecode</a>
CRYPT_HASH_SHA1	Расчёт HASH SHA1	<a href="#">CryptEncode</a> , <a href="#">CryptDecode</a>
CRYPT_HASH_SHA256	Расчёт HASH SHA256	<a href="#">CryptEncode</a> , <a href="#">CryptDecode</a>
DBL_DIG	Количество значимых десятичных знаков	<a href="#">Константы числовых типов</a>
DBL_EPSILON	Наименьшее число для которого выполняется условие $1.0 + DBL\_EPSILON \neq 1.0$	<a href="#">Константы числовых типов</a>
DBL_MANT_DIG	Количество битов в мантиссе	<a href="#">Константы числовых типов</a>
DBL_MAX	Максимальное значение, которое может быть представлено типом double	<a href="#">Константы числовых типов</a>
DBL_MAX_10_EXP	Максимальное десятичное значение степени экспоненты	<a href="#">Константы числовых типов</a>
DBL_MAX_EXP	Максимальное двоичное значение степени экспоненты	<a href="#">Константы числовых типов</a>
DBL_MIN	Минимальное положительное значение, которое может быть представлено типом double	<a href="#">Константы числовых типов</a>
DBL_MIN_10_EXP	Минимальное десятичное значение степени экспоненты	<a href="#">Константы числовых типов</a>
DBL_MIN_EXP	Минимальное двоичное значение степени экспоненты	<a href="#">Константы числовых типов</a>
DEAL_COMMENT	Комментарий к сделке	<a href="#">HistoryDealGetString</a>
DEAL_COMMISSION	Комиссия по сделке	<a href="#">HistoryDealGetDouble</a>
DEAL_ENTRY	Направление сделки - вход в рынок, выход из рынка или разворот	<a href="#">HistoryDealGetInteger</a>

DEAL_ENTRY_IN	Вход в рынок	<a href="#">HistoryDealGetInteger</a>
DEAL_ENTRY_INOUT	Разворот	<a href="#">HistoryDealGetInteger</a>
DEAL_ENTRY_OUT	Выход из рынка	<a href="#">HistoryDealGetInteger</a>
DEAL_MAGIC	Magic number для сделки (смотри <a href="#">ORDER_MAGIC</a> )	<a href="#">HistoryDealGetInteger</a>
DEAL_ORDER	<a href="#">Ордер</a> , на основание которого выполнена сделка	<a href="#">HistoryDealGetInteger</a>
DEAL_POSITION_ID	<a href="#">Идентификатор позиции</a> , в открытии, изменении или закрытии которой участвовала эта сделка. Каждая позиция имеет уникальный идентификатор, который присваивается всем сделкам, совершенным на инструменте в течение всей жизни позиции.	<a href="#">HistoryDealGetInteger</a>
DEAL_PRICE	Цена сделки	<a href="#">HistoryDealGetDouble</a>
DEAL_PROFIT	Финансовый результат сделки	<a href="#">HistoryDealGetDouble</a>
DEAL_SWAP	Накопленный своп при закрытии	<a href="#">HistoryDealGetDouble</a>
DEAL_SYMBOL	Имя символа, по которому произведена сделка	<a href="#">HistoryDealGetString</a>
DEAL_TIME	Время совершения сделки	<a href="#">HistoryDealGetInteger</a>
DEAL_TIME_MSC	Время совершения сделки в миллисекундах с 01.01.1970	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE	Тип сделки	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_BALANCE	Начисление баланса	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_BONUS	Перечисление бонусов	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_BUY	Покупка	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_BUY_CANCELED	Отмененная сделка покупки. Возможная ситуация, когда ранее совершенная сделка на покупку отменяется. В таком случае тип ранее совершенной сделки (DEAL_TYPE_BUY) меняется на DEAL_TYPE_BUY_CANCELED, а	<a href="#">HistoryDealGetInteger</a>

	ее прибыль/убыток обнуляется. Ранее полученная прибыль/убыток начисляется/ списывается со счета отдельной балансовой операцией	
DEAL_TYPE_CHARGE	Дополнительные сборы	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_COMMISSION	Дополнительные комиссии	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_COMMISSION_ANONY_DAILY	Агентская комиссия, начисляемая в конце торгового дня	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_COMMISSION_ANONY_MONTHLY	Агентская комиссия, начисляемая в конце месяца	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_COMMISSION_DAILY	Комиссия, начисляемая в конце торгового дня	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_COMMISSION_MONTHLY	Комиссия, начисляемая в конце месяца	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_CORRECTION	Корректирующая запись	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_CREDIT	Начисление кредита	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_INTEREST	Начисления процентов на свободные средства	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_SELL	Продажа	<a href="#">HistoryDealGetInteger</a>
DEAL_TYPE_SELL_CANCELED	Отмененная сделка продажи. Возможная ситуация, когда ранее совершенная сделка на продажу отменяется. В таком случае тип ранее совершенной сделки (DEAL_TYPE_SELL) меняется на DEAL_TYPE_SELL_CANCELED, а ее прибыль/убыток обнуляется. Ранее полученная прибыль/убыток начисляется/ списывается со счета отдельной балансовой операцией	<a href="#">HistoryDealGetInteger</a>
DEAL_VOLUME	Объем сделки	<a href="#">HistoryDealGetDouble</a>
<a href="#">DRAW_ARROW</a>	Отрисовка стрелками	<a href="#">Стили рисования</a>
<a href="#">DRAW_BARS</a>	Отображение в виде баров	<a href="#">Стили рисования</a>
<a href="#">DRAW_CANDLES</a>	Отображение в виде свечей	<a href="#">Стили рисования</a>

<a href="#">DRAW_COLOR_ARROW</a>	Отрисовка разноцветными стрелками	<a href="#">Стили рисования</a>
<a href="#">DRAW_COLOR_BARS</a>	Разноцветные бары	<a href="#">Стили рисования</a>
<a href="#">DRAW_COLOR_CANDLES</a>	Разноцветные свечи	<a href="#">Стили рисования</a>
<a href="#">DRAW_COLOR_HISTOGRAM</a>	Разноцветная гистограмма от нулевой линии	<a href="#">Стили рисования</a>
<a href="#">DRAW_COLOR_HISTOGRAM2</a>	Разноцветная гистограмма на двух индикаторных буферах	<a href="#">Стили рисования</a>
<a href="#">DRAW_COLOR_LINE</a>	Разноцветная линия	<a href="#">Стили рисования</a>
<a href="#">DRAW_COLOR_SECTION</a>	Разноцветные отрезки	<a href="#">Стили рисования</a>
<a href="#">DRAW_COLOR_ZIGZAG</a>	Разноцветный ZigZag	<a href="#">Стили рисования</a>
<a href="#">DRAW_FILLING</a>	Цветовая заливка между двумя уровнями	<a href="#">Стили рисования</a>
<a href="#">DRAW_HISTOGRAM</a>	Гистограмма от нулевой линии	<a href="#">Стили рисования</a>
<a href="#">DRAW_HISTOGRAM2</a>	Гистограмма на двух индикаторных буферах	<a href="#">Стили рисования</a>
<a href="#">DRAW_LINE</a>	Линия	<a href="#">Стили рисования</a>
<a href="#">DRAW_NONE</a>	Не отрисовывается	<a href="#">Стили рисования</a>
<a href="#">DRAW_SECTION</a>	Отрезки	<a href="#">Стили рисования</a>
<a href="#">DRAW_ZIGZAG</a>	Стиль Zigzag допускает вертикальные отрезки на баре	<a href="#">Стили рисования</a>
<a href="#">ELLIOTT_CYCLE</a>	Цикл (Cycle)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">ELLIOTT_GRAND_SUPERCYCLE</a>	Главный Суперцикл (Grand Supercycle)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">ELLIOTT_INTERMEDIATE</a>	Промежуточное звено (Intermediate)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">ELLIOTT_MINOR</a>	Второстепенный цикл (Minor)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">ELLIOTT_MINUETTE</a>	Секунда (Minuette)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">ELLIOTT_MINUTE</a>	Минута (Minute)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">ELLIOTT_PRIMARY</a>	Первичный цикл (Primary)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>

ELLIOTT_SUBMINUETTE	Субсекунда (Subminuette)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
ELLIOTT_SUPERCYCLE	Суперцикл (Supercycle)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
EMPTY_VALUE	Пустое значение в индикаторном буфере	<a href="#">Прочие константы</a>
ERR_ACCOUNT_WRONG_PROPERTY	Ошибочный идентификатор свойства счета	<a href="#">GetLastError</a>
ERR_ARRAY_BAD_SIZE	Запрашиваемый размер массива превышает 2 гигабайта	<a href="#">GetLastError</a>
ERR_ARRAY_RESIZE_ERROR	Недостаточно памяти для перераспределения массива либо попытка изменения размера статического массива	<a href="#">GetLastError</a>
ERR_BOOKS_CANNOT_ADD	Стакан цен не может быть добавлен	<a href="#">GetLastError</a>
ERR_BOOKS_CANNOT_DELETE	Стакан цен не может быть удален	<a href="#">GetLastError</a>
ERR_BOOKS_CANNOT_GET	Данные стакана цен не могут быть получены	<a href="#">GetLastError</a>
ERR_BOOKS_CANNOT_SUBSCRIBE	Ошибка при подписке на получение новых данных стакана цен	<a href="#">GetLastError</a>
ERR_BUFFERS_NO_MEMORY	Недостаточно памяти для распределения индикаторных буферов	<a href="#">GetLastError</a>
ERR_BUFFERS_WRONG_INDEX	Ошибочный индекс своего индикаторного буфера	<a href="#">GetLastError</a>
ERR_CANNOT_CLEAN_DIRECTORY	Не удалось очистить директорию (возможно, один или несколько файлов заблокированы и операция удаления не удалась)	<a href="#">GetLastError</a>
ERR_CANNOT_DELETE_DIRECTORY	Директория не может быть удалена	<a href="#">GetLastError</a>
ERR_CANNOT_DELETE_FILE	Ошибка удаления файла	<a href="#">GetLastError</a>
ERR_CANNOT_OPEN_FILE	Ошибка открытия файла	<a href="#">GetLastError</a>

ERR_CHAR_ARRAY_ONLY	Должен быть массив типа char	<a href="#">GetLastError</a>
ERR_CHART_CANNOT_CHANGE	Ошибка при изменении для графика символа и периода	<a href="#">GetLastError</a>
ERR_CHART_CANNOT_CREATE_TIMER	Ошибка при создании таймера	<a href="#">GetLastError</a>
ERR_CHART_CANNOT_OPEN	Ошибка открытия графика	<a href="#">GetLastError</a>
ERR_CHART_INDICATOR_CANNOT_ADD	Ошибка при добавлении индикатора на график	<a href="#">GetLastError</a>
ERR_CHART_INDICATOR_CANNOT_DEL	Ошибка при удалении индикатора с графика	<a href="#">GetLastError</a>
ERR_CHART_INDICATOR_NOT_FOUND	Индикатор не найден на указанном графике	<a href="#">GetLastError</a>
ERR_CHART_NAVIGATE_FAILED	Ошибка навигации по графику	<a href="#">GetLastError</a>
ERR_CHART_NO_EXPERT	У графика нет эксперта, который мог бы обработать событие	<a href="#">GetLastError</a>
ERR_CHART_NO_REPLY	График не отвечает	<a href="#">GetLastError</a>
ERR_CHART_NOT_FOUND	График не найден	<a href="#">GetLastError</a>
ERR_CHART_SCREENSHOT_FAILED	Ошибка при создании скриншота	<a href="#">GetLastError</a>
ERR_CHART_TEMPLATE_FAILED	Ошибка при применении шаблона	<a href="#">GetLastError</a>
ERR_CHART_WINDOW_NOT_FOUND	Подокно, содержащее указанный индикатор, не найдено	<a href="#">GetLastError</a>
ERR_CHART_WRONG_ID	Ошибочный идентификатор графика	<a href="#">GetLastError</a>
ERR_CHART_WRONG_PARAMETER	Ошибочное значение параметра для <a href="#">функции по работе с графиком</a>	<a href="#">GetLastError</a>
ERR_CHART_WRONG_PROPERTY	Ошибочный идентификатор свойства графика	<a href="#">GetLastError</a>
ERR_CUSTOM_WRONG PROPERTY	Ошибочный идентификатор свойства пользовательского индикатора	<a href="#">GetLastError</a>
ERR_DIRECTORY_NOT_EXIST	Директория не существует	<a href="#">GetLastError</a>

ERR_DOUBLE_ARRAY_ONLY	Должен быть массив типа double	<a href="#">GetLastError</a>
ERR_FILE_BINSTRINGSIZE	Должен быть указан размер строки, так как файл открыт как бинарный	<a href="#">GetLastError</a>
ERR_FILE_CACHEBUFFER_ERROR	Недостаточно памяти для кеша чтения	<a href="#">GetLastError</a>
ERR_FILE_CANNOT_REWRITE	Файл не может быть переписан	<a href="#">GetLastError</a>
ERR_FILE_IS_DIRECTORY	Это не файл, а директория	<a href="#">GetLastError</a>
ERR_FILE_ISNOT_DIRECTORY	Это файл, а не директория	<a href="#">GetLastError</a>
ERR_FILE_NOT_EXIST	Файл не существует	<a href="#">GetLastError</a>
ERR_FILE_NOTBIN	Файл должен быть открыт как бинарный	<a href="#">GetLastError</a>
ERR_FILE_NOTCSV	Файл должен быть открыт как CSV	<a href="#">GetLastError</a>
ERR_FILE_NOTTOREAD	Файл должен быть открыт для чтения	<a href="#">GetLastError</a>
ERR_FILE_NOTTOWRITE	Файл должен быть открыт для записи	<a href="#">GetLastError</a>
ERR_FILE_NOTTXT	Файл должен быть открыт как текстовый	<a href="#">GetLastError</a>
ERR_FILE_NOTTXTORCSV	Файл должен быть открыт как текстовый или CSV	<a href="#">GetLastError</a>
ERR_FILE_READERROR	Ошибка чтения файла	<a href="#">GetLastError</a>
ERR_FILE_WRITEERROR	Не удалось записать ресурс в файл	<a href="#">GetLastError</a>
ERR_FLOAT_ARRAY_ONLY	Должен быть массив типа float	<a href="#">GetLastError</a>
ERR_FTP_SEND_FAILED	Не удалось отправить файл по ftp	<a href="#">GetLastError</a>
ERR_FUNCTION_NOT_ALLOWED	Системная функция не разрешена для вызова	<a href="#">GetLastError</a>
ERR_GLOBALVARIABLE_EXISTS	Глобальная переменная клиентского терминала с таким именем уже существует	<a href="#">GetLastError</a>
ERR_GLOBALVARIABLE_NOT_FOUND	Глобальная переменная клиентского терминала не	<a href="#">GetLastError</a>

	найдена	
ERR_HISTORY_NOT_FOUND	Запрашиваемая история не найдена	<a href="#">GetLastError</a>
ERR_HISTORY_WRONG_PROPERTY	Ошибочный идентификатор свойства истории	<a href="#">GetLastError</a>
ERR_INCOMPATIBLE_ARRAYS	Копирование несовместимых массивов. Строковый массив может быть скопирован только в строковый, а числовой массив - в числовой	<a href="#">GetLastError</a>
ERR_INCOMPATIBLE_FILE	Для строковых массивов должен быть текстовый файл, для остальных - бинарный	<a href="#">GetLastError</a>
ERR_INDICATOR_CANNOT_ADD	Ошибка при добавлении индикатора	<a href="#">GetLastError</a>
ERR_INDICATOR_CANNOT_APPLY	Индикатор не может быть применен к другому индикатору	<a href="#">GetLastError</a>
ERR_INDICATOR_CANNOT_CREATE	Индикатор не может быть создан	<a href="#">GetLastError</a>
ERR_INDICATOR_CUSTOM_NAME	Первым параметром в массиве должно быть имя пользовательского индикатора	<a href="#">GetLastError</a>
ERR_INDICATOR_DATA_NOT_FOUND	Запрошенные данные не найдены	<a href="#">GetLastError</a>
ERR_INDICATOR_NO_MEMORY	Недостаточно памяти для добавления индикатора	<a href="#">GetLastError</a>
ERR_INDICATOR_PARAMETER_TYPE	Неправильный тип параметра в массиве при создании индикатора	<a href="#">GetLastError</a>
ERR_INDICATOR_PARAMETERS_MISSING	Отсутствуют параметры при создании индикатора	<a href="#">GetLastError</a>
ERR_INDICATOR_UNKNOWN_SYMBOL	Неизвестный символ	<a href="#">GetLastError</a>
ERR_INDICATOR_WRONG_HANDLE	Ошибочный хэндл индикатора	<a href="#">GetLastError</a>
ERR_INDICATOR_WRONG_INDEX	Ошибочный индекс запрашиваемого индикаторного буфера	<a href="#">GetLastError</a>

ERR_INDICATOR_WRONG_PARAMETERS	Неправильное количество параметров при создании индикатора	<a href="#">GetLastError</a>
ERR_INT_ARRAY_ONLY	Должен быть массив типа int	<a href="#">GetLastError</a>
ERR_INTERNAL_ERROR	Неожиданная внутренняя ошибка	<a href="#">GetLastError</a>
ERR_INVALID_ARRAY	Массив неподходящего типа, неподходящего размера или испорченный объект динамического массива	<a href="#">GetLastError</a>
ERR_INVALID_DATETIME	Неправильное значение даты и/или времени	<a href="#">GetLastError</a>
ERR_INVALID_FILEHANDLE	Файл с таким хэндлом уже был закрыт, либо не открывался вообще	<a href="#">GetLastError</a>
ERR_INVALID_PARAMETER	Ошибочный параметр при вызове системной функции	<a href="#">GetLastError</a>
ERR_INVALID_POINTER	Ошибочный указатель	<a href="#">GetLastError</a>
ERR_INVALID_POINTER_TYPE	Ошибочный тип указателя	<a href="#">GetLastError</a>
ERR_LONG_ARRAY_ONLY	Должен быть массив типа long	<a href="#">GetLastError</a>
ERR_MAIL_SEND_FAILED	Не удалось отправить письмо	<a href="#">GetLastError</a>
ERR_MARKET_LASTTIME_UNKNOWN	Время последнего тика неизвестно (тиков не было)	<a href="#">GetLastError</a>
ERR_MARKET_NOT_SELECTED	Символ не выбран в MarketWatch	<a href="#">GetLastError</a>
ERR_MARKET_SELECT_ERROR	Ошибка добавления или удаления символа в MarketWatch	<a href="#">GetLastError</a>
ERR_MARKET_UNKNOWN_SYMBOL	Неизвестный символ	<a href="#">GetLastError</a>
ERR_MARKET_WRONG_PROPERTY	Ошибочный идентификатор свойства символа	<a href="#">GetLastError</a>
ERR_MQL5_WRONG_PROPERTY	Ошибочный идентификатор свойства программы	<a href="#">GetLastError</a>
ERR_NO_STRING_DATE	В строке нет даты	<a href="#">GetLastError</a>
ERR_NOT_ENOUGH_MEMORY	Недостаточно памяти для выполнения системной функции	<a href="#">GetLastError</a>

ERR_NOTIFICATION_SEND_FAILED	Не удалось отправить <a href="#">уведомление</a>	<a href="#">GetLastError</a>
ERR_NOTIFICATION_TOO_FREQUENT	Слишком частая отправка уведомлений	<a href="#">GetLastError</a>
ERR_NOTIFICATION_WRONG_PARAMETER	Неверный параметр для отправки уведомления - в функцию <a href="#">SendNotification()</a> передали пустую строку или <a href="#">NULL</a>	<a href="#">GetLastError</a>
ERR_NOTIFICATION_WRONG_SETTINGS	Неверные настройки уведомлений в терминале (не указан ID или не выставлено разрешение)	<a href="#">GetLastError</a>
ERR_NOTINITIALIZED_STRING	Неинициализированная строка	<a href="#">GetLastError</a>
ERR_NUMBER_ARRAYS_ONLY	Должен быть числовой массив	<a href="#">GetLastError</a>
ERR_OBJECT_ERROR	Ошибка при работе с графическим объектом	<a href="#">GetLastError</a>
ERR_OBJECT_GETDATE_FAILED	Невозможно получить дату, соответствующую значению	<a href="#">GetLastError</a>
ERR_OBJECT_GETVALUE_FAILED	Невозможно получить значение, соответствующее дате	<a href="#">GetLastError</a>
ERR_OBJECT_NOT_FOUND	Графический объект не найден	<a href="#">GetLastError</a>
ERR_OBJECT_WRONG_PROPERTY	Ошибочный идентификатор свойства графического объекта	<a href="#">GetLastError</a>
ERR_ONEDIM_ARRAYS_ONLY	Должен быть одномерный массив	<a href="#">GetLastError</a>
ERR_OPENCL_BUFFER_CREATE	Ошибка создания <a href="#">буфера OpenCL</a>	<a href="#">GetLastError</a>
ERR_OPENCL_CONTEXT_CREATE	Ошибка при создании <a href="#">контекста OpenCL</a>	<a href="#">GetLastError</a>
ERR_OPENCL_EXECUTE	Ошибка выполнения <a href="#">программы OpenCL</a>	<a href="#">GetLastError</a>
ERR_OPENCL_INTERNAL	Внутренняя ошибка при <a href="#">выполнении OpenCL</a>	<a href="#">GetLastError</a>

ERR_OPENCL_INVALID_HANDLE	Неправильный <a href="#">хэндл OpenCL</a>	<a href="#">GetLastError</a>
ERR_OPENCL_KERNEL_CREATE	Ошибка создания кернел - <a href="#">точки входа OpenCL</a>	<a href="#">GetLastError</a>
ERR_OPENCL_NOT_SUPPORTED	<a href="#">Функции OpenCL</a> на данном компьютере не поддерживаются	<a href="#">GetLastError</a>
ERR_OPENCL_PROGRAM_CREATE	Ошибка при <a href="#">компиляции программы OpenCL</a>	<a href="#">GetLastError</a>
ERR_OPENCL_QUEUE_CREATE	Ошибка создания очереди выполнения в OpenCL	<a href="#">GetLastError</a>
ERR_OPENCL_SET_KERNEL_PARAMETER	Ошибка при <a href="#">установке параметров</a> для кернел OpenCL (точки входа в программу OpenCL)	<a href="#">GetLastError</a>
ERR_OPENCL_TOO_LONG_KERNEL_NAME	Слишком длинное имя точки входа ( <a href="#">кернел OpenCL</a> )	<a href="#">GetLastError</a>
ERR_OPENCL_WRONG_BUFFER_OFFSET	Неверное смещение в буфере OpenCL	<a href="#">GetLastError</a>
ERR_OPENCL_WRONG_BUFFER_SIZE	Неверный размер буфера OpenCL	<a href="#">GetLastError</a>
ERR_PLAY_SOUND_FAILED	Не удалось воспроизвести звук	<a href="#">GetLastError</a>
ERR_RESOURCE_NAME_DUPLICATED	Совпадение имени <a href="#">динамического</a> и <a href="#">статического</a> ресурсов	<a href="#">GetLastError</a>
ERR_RESOURCE_NAME_IS_TOO_LONG	Имя ресурса превышает 63 символа	<a href="#">GetLastError</a>
ERR_RESOURCE_NOT_FOUND	Ресурс с таким именем в EX5 не найден	<a href="#">GetLastError</a>
ERR_RESOURCE_UNSUPPORTED_TYPE	Неподдерживаемый тип ресурса или размер более 16 МВ	<a href="#">GetLastError</a>
ERR_SERIES_ARRAY	Таймсериya не может быть использована	<a href="#">GetLastError</a>
ERR_SHORT_ARRAY_ONLY	Должен быть массив типа short	<a href="#">GetLastError</a>
ERR_SMALL_ARRAY	Слишком маленький массив, стартовая позиция за пределами массива	<a href="#">GetLastError</a>

ERR_SMALL_ASSERIES_ARRAY	Приемный массив объявлен как AS_SERIES, и он недостаточного размера	<a href="#">GetLastError</a>
ERR_STRING_OUT_OF_MEMORY	Недостаточно памяти для строки	<a href="#">GetLastError</a>
ERR_STRING_RESIZE_ERROR	Недостаточно памяти для перераспределения строки	<a href="#">GetLastError</a>
ERR_STRING_SMALL_LEN	Длина строки меньше, чем ожидалось	<a href="#">GetLastError</a>
ERR_STRING_TIME_ERROR	Ошибка преобразования строки в дату	<a href="#">GetLastError</a>
ERR_STRING_TOOBIGNUMBER	Слишком большое число, больше, чем ULONG_MAX	<a href="#">GetLastError</a>
ERR_STRING_UNKNOWNTYPE	Неизвестный тип данных при конвертации в строку	<a href="#">GetLastError</a>
ERR_STRING_ZEROADDED	К концу строки добавлен 0, бесполезная операция	<a href="#">GetLastError</a>
ERR_STRINGPOS_OUTOFRANGE	Позиция за пределами строки	<a href="#">GetLastError</a>
ERR_STRUCT_WITHOBJECTS_ORCLASS	Структура содержит объекты строк и/или динамических массивов и/или структуры с такими объектами и/или классы	<a href="#">GetLastError</a>
ERR_SUCCESS	Операция выполнена успешно	<a href="#">GetLastError</a>
ERR_TERMINAL_WRONG_PROPERTY	Ошибочный идентификатор свойства терминала	<a href="#">GetLastError</a>
ERR_TOO_LONG_FILENAME	Слишком длинное имя файла	<a href="#">GetLastError</a>
ERR_TOO_MANY_FILES	Не может быть открыто одновременно более 64 файлов	<a href="#">GetLastError</a>
ERR_TOO_MANY_FORMATTERS	Форматных спецификаторов больше, чем параметров	<a href="#">GetLastError</a>
ERR_TOO_MANY_PARAMETERS	Параметров больше, чем форматных спецификаторов	<a href="#">GetLastError</a>
ERR_TRADE DEAL NOT FOUND	Сделка не найдена	<a href="#">GetLastError</a>
ERR_TRADE_DISABLED	Торговля для эксперта запрещена	<a href="#">GetLastError</a>

ERR_TRADE_ORDER_NOT_FOUND	Ордер не найден	<a href="#">GetLastError</a>
ERR_TRADE_POSITION_NOT_FOUND	Позиция не найдена	<a href="#">GetLastError</a>
ERR_TRADE_SEND_FAILED	Не удалось отправить торговый запрос	<a href="#">GetLastError</a>
ERR_TRADE_WRONG_PROPERTY	Ошибочный идентификатор свойства торговли	<a href="#">GetLastError</a>
ERR_USER_ERROR_FIRST	С этого кода начинаются ошибки, <a href="#">задаваемые пользователем</a>	<a href="#">GetLastError</a>
ERR_WEBREQUEST_CONNECT_FAILED	Не удалось подключиться к указанному URL	<a href="#">GetLastError</a>
ERR_WEBREQUEST_INVALID_ADDRESS	URL не прошел проверку	<a href="#">GetLastError</a>
ERR_WEBREQUEST_REQUEST_FAILED	Ошибка в результате выполнения HTTP запроса	<a href="#">GetLastError</a>
ERR_WEBREQUEST_TIMEOUT	Превышен таймаут получения данных	<a href="#">GetLastError</a>
ERR_WRONG_DIRECTORYNAME	Ошибочное имя директории	<a href="#">GetLastError</a>
ERR_WRONG_FILEHANDLE	Ошибочный хэндл файла	<a href="#">GetLastError</a>
ERR_WRONG_FILENAME	Недопустимое имя файла	<a href="#">GetLastError</a>
ERR_WRONG_FORMATSTRING	Ошибочная форматная строка	<a href="#">GetLastError</a>
ERR_WRONG_INTERNAL_PARAMETER	Ошибочный параметр при внутреннем вызове функции клиентского терминала	<a href="#">GetLastError</a>
ERR_WRONG_STRING_DATE	В строке ошибочная дата	<a href="#">GetLastError</a>
ERR_WRONG_STRING_OBJECT	Испорченный объект строки	<a href="#">GetLastError</a>
ERR_WRONG_STRING_PARAMETER	Испорченный параметр типа string	<a href="#">GetLastError</a>
ERR_WRONG_STRING_TIME	В строке ошибочное время	<a href="#">GetLastError</a>
ERR_ZEROSIZE_ARRAY	Массив нулевой длины	<a href="#">GetLastError</a>
FILE_ACCESS_DATE	Дата последнего доступа к файлу	<a href="#">FileGetInteger</a>
FILE_ANSI	Строки типа ANSI (однобайтовые символы).	<a href="#">FileOpen</a>

	Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> )	
FILE_BIN	Двоичный режим чтения-записи (без преобразования из строки и в строку). Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> )	<a href="#">FileOpen</a>
FILE_COMMON	Расположение файла в общей папке всех клиентских терминалов \Terminal\Common\Files. Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> ), копировании файлов ( <a href="#">FileCopy()</a> ), <a href="#">FileMove()</a> и проверке существования файлов ( <a href="#">FileIsExist()</a> )	<a href="#">FileOpen</a> , <a href="#">FileCopy</a> , <a href="#">FileMove</a> , <a href="#">FileIsExist</a>
FILE_CREATE_DATE	Дата создания	<a href="#">FileGetInteger</a>
FILE_CSV	Файл типа csv (все записанные элементы преобразуются к строкам соответствующего типа, unicode или ansi, и разделяются разделителем). Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> )	<a href="#">FileOpen</a>
FILE_END	Получение признака конца файла	<a href="#">FileGetInteger</a>
FILE_EXISTS	Проверка на существование	<a href="#">FileGetInteger</a>
FILE_IS_ANSI	Файл открыт как ANSI (смотри <a href="#">FILE_ANSI</a> )	<a href="#">FileGetInteger</a>
FILE_IS_BINARY	Файл открыт как бинарный (смотри <a href="#">FILE_BIN</a> )	<a href="#">FileGetInteger</a>
FILE_IS_COMMON	Файл открыт в общей папке всех клиентских терминалов (смотри <a href="#">FILE_COMMON</a> )	<a href="#">FileGetInteger</a>
FILE_IS_CSV	Файл открыт как CSV (смотри <a href="#">FILE_CSV</a> )	<a href="#">FileGetInteger</a>
FILE_IS_READABLE	Файл открыт с возможностью чтения (смотри <a href="#">FILE_READ</a> )	<a href="#">FileGetInteger</a>
FILE_IS_TEXT	Файл открыт как текстовый (смотри <a href="#">FILE_TXT</a> )	<a href="#">FileGetInteger</a>

FILE_IS_WRITABLE	Файл открыт с возможностью записи (смотри <a href="#">FILE_WRITE</a> )	<a href="#">FileGetInteger</a>
FILE_LINE_END	Получение признака конца строки	<a href="#">FileGetInteger</a>
FILE MODIFY_DATE	Дата последнего изменения	<a href="#">FileGetInteger</a>
FILE_POSITION	Позиция указателя в файле	<a href="#">FileGetInteger</a>
FILE_READ	Файл открывается для чтения. Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> ). При открытии файла обязательно должен быть указан флаг FILE_WRITE и/или флаг FILE_READ	<a href="#">FileOpen</a>
FILE_REWRITE	Возможность перезаписывания файла функциями <a href="#">FileCopy()</a> и <a href="#">FileMove()</a> . Файл должен существовать или открываться для записи. В противном случае файл открыт не будет	<a href="#">FileCopy</a> , <a href="#">FileMove</a>
FILE_SHARE_READ	Совместный доступ по чтению со стороны нескольких программ. Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> ), но не заменяет при открытии файла необходимости указать FILE_WRITE и/или флаг FILE_READ	<a href="#">FileOpen</a>
FILE_SHARE_WRITE	Совместный доступ по записи со стороны нескольких программ. Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> ), но не заменяет при открытии файла необходимости указать FILE_WRITE и/или флаг FILE_READ	<a href="#">FileOpen</a>
FILE_SIZE	Размер файла в байтах	<a href="#">FileGetInteger</a>
FILE_TXT	Простой текстовый файл (тот же csv, однако разделятель не принимается во внимание). Флаг	<a href="#">FileOpen</a>

	используется при открытии файлов ( <a href="#">FileOpen()</a> )	
FILE_UNICODE	Строки типа UNICODE (двухбайтовые символы). Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> )	<a href="#">FileOpen</a>
FILE_WRITE	Файл открывается для записи. Флаг используется при открытии файлов ( <a href="#">FileOpen()</a> ). При открытии файла обязательно должен быть указан флаг FILE_WRITE и/или флаг FILE_READ	<a href="#">FileOpen</a>
FLT_DIG	Количество значимых десятичных знаков	<a href="#">Константы числовых типов</a>
FLT_EPSILON	Наименьшее число для которого выполняется условие $1.0 + \text{FLT\_EPSILON} \neq 1.0$	<a href="#">Константы числовых типов</a>
FLT_MANT_DIG	Количество битов в мантиссе	<a href="#">Константы числовых типов</a>
FLT_MAX	Максимальное значение, которое может быть представлено типом float	<a href="#">Константы числовых типов</a>
FLT_MAX_10_EXP	Максимальное десятичное значение степени экспоненты	<a href="#">Константы числовых типов</a>
FLT_MAX_EXP	Максимальное двоичное значение степени экспоненты	<a href="#">Константы числовых типов</a>
FLT_MIN	Минимальное положительное значение, которое может быть представлено типом float	<a href="#">Константы числовых типов</a>
FLT_MIN_10_EXP	Минимальное десятичное значение степени экспоненты	<a href="#">Константы числовых типов</a>
FLT_MIN_EXP	Минимальное двоичное значение степени экспоненты	<a href="#">Константы числовых типов</a>
FRIDAY	Пятница	<a href="#">SymbolInfoInteger</a> , <a href="#">SymbolInfoSessionQuote</a> , <a href="#">SymbolInfoSessionTrade</a>
GANN_DOWN_TREND	Линия соответствует нисходящему тренду	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
GANN_UP_TREND	Линия соответствует восходящему тренду	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>

GATORJAW_LINE	Линия челюстей	<a href="#">Линии индикаторов</a>
GATORLIPS_LINE	Линия губ	<a href="#">Линии индикаторов</a>
GATORTEETH_LINE	Линия зубов	<a href="#">Линии индикаторов</a>
IDABORT	Выбрана кнопка Прервать (Abort)	<a href="#">MessageBox</a>
IDCANCEL	Выбрана кнопка Отмена (Cancel)	<a href="#">MessageBox</a>
IDCONTINUE	Выбрана кнопка Продолжить (Continue)	<a href="#">MessageBox</a>
IDIGNORE	Выбрана кнопка Пропустить (Ignore)	<a href="#">MessageBox</a>
IDNO	Выбрана кнопка Нет (No)	<a href="#">MessageBox</a>
IDOK	Выбрана кнопка OK	<a href="#">MessageBox</a>
IDRETRY	Выбрана кнопка Повтор (Retry)	<a href="#">MessageBox</a>
IDTRYAGAIN	Выбрана кнопка Повторить (Try Again)	<a href="#">MessageBox</a>
IDYES	Выбрана кнопка Да (Yes)	<a href="#">MessageBox</a>
IND_AC	Accelerator Oscillator	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_AD	Accumulation/Distribution	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_ADX	Average Directional Index	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_AXDW	ADX by Welles Wilder	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_ALLIGATOR	Alligator	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_AMA	Adaptive Moving Average	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_AO	Awesome Oscillator	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_ATR	Average True Range	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_BANDS	Bollinger Bands®	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>

IND_BEARS	Bears Power	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_BULLS	Bulls Power	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_BWMFI	Market Facilitation Index	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_CCI	Commodity Channel Index	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_CHAIKIN	Chaikin Oscillator	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_CUSTOM	Custom indicator	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_DEMA	Double Exponential Moving Average	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_DEMARKER	DeMarker	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_ENVELOPES	Envelopes	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_FORCE	Force Index	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_FRACTALS	Fractals	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_FRAMA	Fractal Adaptive Moving Average	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_GATOR	Gator Oscillator	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_ICHIMOKU	Ichimoku Kinko Hyo	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_MA	Moving Average	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_MACD	MACD	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_MFI	Money Flow Index	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_MOMENTUM	Momentum	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_OBV	On Balance Volume	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>

IND_OSMA	OsMA	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_RSI	Relative Strength Index	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_RVI	Relative Vigor Index	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_SAR	Parabolic SAR	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_STDDEV	Standard Deviation	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_STOCHASTIC	Stochastic Oscillator	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_TEMA	Triple Exponential Moving Average	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_TRIX	Triple Exponential Moving Averages Oscillator	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_VIDYA	Variable Index Dynamic Average	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_VOLUMES	Volumes	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
IND_WPR	Williams' Percent Range	<a href="#">IndicatorCreate</a> , <a href="#">IndicatorParameters</a>
INDICATOR_CALCULATIONS	Вспомогательные буфера для промежуточных вычислений	<a href="#">SetIndexBuffer</a>
INDICATOR_COLOR_INDEX	Цвета отрисовки	<a href="#">SetIndexBuffer</a>
INDICATOR_DATA	Данные для отрисовки	<a href="#">SetIndexBuffer</a>
INDICATOR_DIGITS	Точность отображения значений индикатора	<a href="#">IndicatorSetInteger</a>
INDICATOR_HEIGHT	Фиксированная высота собственного окна индикатора (команда препроцессора <a href="#">#property indicator_height</a> )	<a href="#">IndicatorSetInteger</a>
INDICATOR_LEVELCOLOR	Цвет линии уровня	<a href="#">IndicatorSetInteger</a>
INDICATOR_LEVELS	Количество уровней на окне индикатора	<a href="#">IndicatorSetInteger</a>
INDICATOR_LEVELSTYLE	Стиль линии уровня	<a href="#">IndicatorSetInteger</a>
INDICATOR_LEVELTEXT	Описание уровня	<a href="#">IndicatorSetString</a>

INDICATOR_LEVELVALUE	Значение уровня	<a href="#">IndicatorSetDouble</a>
INDICATOR_LEVELWIDTH	Толщина линии уровня	<a href="#">IndicatorSetInteger</a>
INDICATOR_MAXIMUM	Максимум окна индикатора	<a href="#">IndicatorSetDouble</a>
INDICATOR_MINIMUM	Минимум окна индикатора	<a href="#">IndicatorSetDouble</a>
INDICATOR_SHORTNAME	Короткое наименование индикатора	<a href="#">IndicatorSetString</a>
INT_MAX	Максимальное значение, которое может быть представлено типом int	<a href="#">Константы числовых типов</a>
INT_MIN	Минимальное значение, которое может быть представлено типом int	<a href="#">Константы числовых типов</a>
INVALID_HANDLE	Некорректный хэндл	<a href="#">Прочие константы</a>
IS_DEBUG_MODE	Признак работы mq5-программы в режиме отладки	<a href="#">Прочие константы</a>
IS_PROFILE_MODE	Признак работы mq5-программы в режиме профилирования	<a href="#">Прочие константы</a>
KIJUNSEN_LINE	Линия Kijun-sen	<a href="#">Линии индикаторов</a>
LICENSE_DEMO	Демо-версия платного продукта из Маркета. Работает только в тестере стратегий	<a href="#">MQLInfoInteger</a>
LICENSE_FREE	Бесплатная неограниченная версия	<a href="#">MQLInfoInteger</a>
LICENSE_FULL	Купленная лицензионная версия допускает не менее 5 активаций. Продавец может увеличить разрешенное число активаций	<a href="#">MQLInfoInteger</a>
LICENSE_TIME	Версия с ограниченной по времени лицензией	<a href="#">MQLInfoInteger</a>
LONG_MAX	Максимальное значение, которое может быть представлено типом long	<a href="#">Константы числовых типов</a>
LONG_MIN	Минимальное значение, которое может быть представлено типом long	<a href="#">Константы числовых типов</a>
LOWER_BAND	Нижняя граница	<a href="#">Линии индикаторов</a>

LOWER_HISTOGRAM	Нижняя гистограмма	<a href="#">Линии индикаторов</a>
LOWER_LINE	Нижняя линия	<a href="#">Линии индикаторов</a>
M_1_PI	1/pi	<a href="#">Математические константы</a>
M_2_PI	2/pi	<a href="#">Математические константы</a>
M_2_SQRTPI	2/sqrt(pi)	<a href="#">Математические константы</a>
M_E	e	<a href="#">Математические константы</a>
M_LN10	ln(10)	<a href="#">Математические константы</a>
M_LN2	ln(2)	<a href="#">Математические константы</a>
M_LOG10E	log10(e)	<a href="#">Математические константы</a>
M_LOG2E	log2(e)	<a href="#">Математические константы</a>
M_PI	pi	<a href="#">Математические константы</a>
M_PI_2	pi/2	<a href="#">Математические константы</a>
M_PI_4	pi/4	<a href="#">Математические константы</a>
M_SQRT1_2	1/sqrt(2)	<a href="#">Математические константы</a>
M_SQRT2	sqrt(2)	<a href="#">Математические константы</a>
MAIN_LINE	Основная линия	<a href="#">Линии индикаторов</a>
MB_ABORTRETRYIGNORE	Окно сообщения содержит три кнопки: Abort, Retry и Ignore	<a href="#">MessageBox</a>
MB_CANCELTRYCONTINUE	Окно сообщения содержит три кнопки: Cancel, Try Again, Continue	<a href="#">MessageBox</a>
MB_DEFBUTTON1	Первая кнопка MB_DEFBUTTON1 - кнопка выбрана по умолчанию, если MB_DEFBUTTON2, MB_DEFBUTTON3, MB_DEFBUTTON4 или не определены	<a href="#">MessageBox</a>
MB_DEFBUTTON2	Вторая кнопка - кнопка по умолчанию	<a href="#">MessageBox</a>
MB_DEFBUTTON3	Третья кнопка - кнопка по умолчанию	<a href="#">MessageBox</a>
MB_DEFBUTTON4	Четвертая кнопка - кнопка по умолчанию	<a href="#">MessageBox</a>
MB_ICONEXCLAMATION, MB_ICONWARNING	Изображение восклицательного знака	<a href="#">MessageBox</a>

MB_ICONINFORMATION, MB_ICONASTERISK	Изображение, состоящее из строчного знака і в круге	<a href="#">MessageBox</a>
MB_ICONQUESTION	Изображение вопросительного знака	<a href="#">MessageBox</a>
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	Изображение знака STOP	<a href="#">MessageBox</a>
MB_OK	Окно сообщения содержит одну кнопку: OK. По умолчанию	<a href="#">MessageBox</a>
MB_OKCANCEL	Окно сообщения содержит две кнопки: OK и Cancel	<a href="#">MessageBox</a>
MB_RETRYCANCEL	Окно сообщения содержит две кнопки: Retry и Cancel	<a href="#">MessageBox</a>
MB_YESNO	Окно сообщения содержит две кнопки: Yes и No	<a href="#">MessageBox</a>
MB_YESNOCANCEL	Окно сообщения содержит три кнопки: Yes, No и Cancel	<a href="#">MessageBox</a>
MINUSDI_LINE	Линия -DI	<a href="#">Линии индикаторов</a>
MODE_EMA	Экспоненциальное усреднение	<a href="#">Методы скользящих</a>
MODE_LWMA	Линейно-взвешенное усреднение	<a href="#">Методы скользящих</a>
MODE_SMA	Простое усреднение	<a href="#">Методы скользящих</a>
MODE_SMMA	Сглаженное усреднение	<a href="#">Методы скользящих</a>
MONDAY	Понедельник	<a href="#">SymbolInfoInteger</a> , <a href="#">SymbolInfoSessionQuote</a> , <a href="#">SymbolInfoSessionTrade</a>
MQL_DEBUG	Признак работы запущенной программы в режиме отладки	<a href="#">MQLInfoInteger</a>
MQL_DLLS_ALLOWED	Разрешение на использование DLL для данной запущенной программы	<a href="#">MQLInfoInteger</a>
MQL_FRAME_MODE	Признак работы запущенного эксперта на графике в режиме сбора фреймов результатов оптимизации	<a href="#">MQLInfoInteger</a>

<code>MQL_LICENSE_TYPE</code>	Тип лицензии модуля EX5. Лицензия относится именно к тому модулю EX5, из которого делается запрос с помощью <code>MQLInfoInteger(MQL_LICENSE_TYPE)</code> .	<a href="#">MQLInfoInteger</a>
<code>MQL_MEMORY_LIMIT</code>	Максимально возможный объём динамической памяти для MQL5-программы в МБ	<a href="#">MQLInfoInteger</a>
<code>MQL_MEMORY_USED</code>	Размер использованной памяти MQL5-программой в МБ	<a href="#">MQLInfoInteger</a>
<code>MQL_OPTIMIZATION</code>	Признак работы запущенной программы в процессе оптимизации	<a href="#">MQLInfoInteger</a>
<code>MQL_PROFILER</code>	Признак работы запущенной программы в режиме профилирования кода	<a href="#">MQLInfoInteger</a>
<code>MQL_PROGRAM_NAME</code>	Имя запущенной MQL5-программы	<a href="#">MQLInfoString</a>
<code>MQL_PROGRAM_PATH</code>	Путь для данной запущенной программы	<a href="#">MQLInfoString</a>
<code>MQL_PROGRAM_TYPE</code>	Тип mql5-программы	<a href="#">MQLInfoInteger</a>
<code>MQL_SIGNALS_ALLOWED</code>	Разрешение на работу с сигналами данной запущенной программы	<a href="#">MQLInfoInteger</a>
<code>MQL_TESTER</code>	Признак работы запущенной программы в тестере	<a href="#">MQLInfoInteger</a>
<code>MQL_TRADE_ALLOWED</code>	<a href="#">Разрешение на торговлю</a> для данной запущенной программы	<a href="#">MQLInfoInteger</a>
<code>MQL_VISUAL_MODE</code>	Признак работы запущенной программы в визуальном режиме тестирования	<a href="#">MQLInfoInteger</a>
<code>NULL</code>	Ноль любого типа	<a href="#">Прочие константы</a>
<code>OBJ_ALL_PERIODS</code>	Объект рисуется на всех таймфреймах	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#"><u>OBJ_ARROW</u></a>	Объект "Стрелка"	<a href="#">Типы объектов</a>
<a href="#"><u>OBJ_ARROW_BUY</u></a>	Знак "Buy"	<a href="#">Типы объектов</a>

<a href="#">OBJ_ARROW_CHECK</a>	Знак "Птичка" (галка)	<a href="#">Типы объектов</a>
<a href="#">OBJ_ARROW_DOWN</a>	Знак "Стрелка вниз"	<a href="#">Типы объектов</a>
<a href="#">OBJ_ARROW_LEFT_PRICE</a>	Левая ценовая метка	<a href="#">Типы объектов</a>
<a href="#">OBJ_ARROW_RIGHT_PRICE</a>	Правая ценовая метка	<a href="#">Типы объектов</a>
<a href="#">OBJ_ARROW_SELL</a>	Знак "Sell"	<a href="#">Типы объектов</a>
<a href="#">OBJ_ARROW_STOP</a>	Знак "Стоп"	<a href="#">Типы объектов</a>
<a href="#">OBJ_ARROW_THUMB_DOWN</a>	Знак "Плохо" (большой палец вниз)	<a href="#">Типы объектов</a>
<a href="#">OBJ_ARROW_THUMB_UP</a>	Знак "Хорошо" (большой палец вверх)	<a href="#">Типы объектов</a>
<a href="#">OBJ_ARROW_UP</a>	Знак "Стрелка вверх"	<a href="#">Типы объектов</a>
<a href="#">OBJ_ARROWED_LINE</a>	Объект "Линия со стрелкой"	<a href="#">Типы объектов</a>
<a href="#">OBJ_BITMAP</a>	Объект "Рисунок"	<a href="#">Типы объектов</a>
<a href="#">OBJ_BITMAP_LABEL</a>	Объект "Графическая метка"	<a href="#">Типы объектов</a>
<a href="#">OBJ_BUTTON</a>	Объект "Кнопка"	<a href="#">Типы объектов</a>
<a href="#">OBJ_CHANNEL</a>	Равноудаленный канал	<a href="#">Типы объектов</a>
<a href="#">OBJ_CHART</a>	Объект "График"	<a href="#">Типы объектов</a>
<a href="#">OBJ_CYCLES</a>	Циклические линии	<a href="#">Типы объектов</a>
<a href="#">OBJ_EDIT</a>	Объект "Поле ввода"	<a href="#">Типы объектов</a>
<a href="#">OBJ_ELLIOTWAVE3</a>	3-волновка Эллиота	<a href="#">Типы объектов</a>
<a href="#">OBJ_ELLIOTWAVE5</a>	5-волновка Эллиота	<a href="#">Типы объектов</a>
<a href="#">OBJ_ELLIPSE</a>	Эллипс	<a href="#">Типы объектов</a>
<a href="#">OBJ_EVENT</a>	Объект "Событие", соответствующий событию в экономическом календаре	<a href="#">Типы объектов</a>
<a href="#">OBJ_EXPANSION</a>	Расширение Фибоначчи	<a href="#">Типы объектов</a>
<a href="#">OBJ_FIBO</a>	Уровни Фибоначчи	<a href="#">Типы объектов</a>
<a href="#">OBJ_FIBOARC</a>	Дуги Фибоначчи	<a href="#">Типы объектов</a>
<a href="#">OBJ_FIBOCHANNEL</a>	Канал Фибоначчи	<a href="#">Типы объектов</a>
<a href="#">OBJ_FIBOFAN</a>	Веер Фибоначчи	<a href="#">Типы объектов</a>
<a href="#">OBJ_FIBOTIMES</a>	Временные зоны Фибоначчи	<a href="#">Типы объектов</a>
<a href="#">OBJ_GANNFAN</a>	Веер Ганна	<a href="#">Типы объектов</a>
<a href="#">OBJ_GANNGRID</a>	Сетка Ганна	<a href="#">Типы объектов</a>

<a href="#">OBJ_GANNLINE</a>	Линия Ганна	<a href="#">Типы объектов</a>
<a href="#">OBJ_HLINE</a>	Горизонтальная линия	<a href="#">Типы объектов</a>
<a href="#">OBJ_LABEL</a>	Объект "Текстовая метка"	<a href="#">Типы объектов</a>
<a href="#">OBJ_NO_PERIODS</a>	Объект не показывается ни на одном таймфрейме	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_D1</a>	Объект рисуется на дневных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_H1</a>	Объект рисуется на 1-часовых графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_H12</a>	Объект рисуется на 12-часовых графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_H2</a>	Объект рисуется на 2-часовых графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_H3</a>	Объект рисуется на 3-часовых графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_H4</a>	Объект рисуется на 4-часовых графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_H6</a>	Объект рисуется на 6-часовых графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_H8</a>	Объект рисуется на 8-часовых графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_M1</a>	Объект рисуется на 1-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_M10</a>	Объект рисуется на 10-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_M12</a>	Объект рисуется на 12-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_M15</a>	Объект рисуется на 15-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_M2</a>	Объект рисуется на 2-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_M20</a>	Объект рисуется на 20-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_M3</a>	Объект рисуется на 3-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PERIOD_M30</a>	Объект рисуется на 30-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>

OBJ_PERIOD_M4	Объект рисуется на 4-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJ_PERIOD_M5	Объект рисуется на 5-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJ_PERIOD_M6	Объект рисуется на 6-минутных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJ_PERIOD_MN1	Объект рисуется на месячных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJ_PERIOD_W1	Объект рисуется на недельных графиках	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
<a href="#">OBJ_PITCHFORK</a>	Вилы Эндрюса	<a href="#">Типы объектов</a>
<a href="#">OBJ_RECTANGLE</a>	Прямоугольник	<a href="#">Типы объектов</a>
<a href="#">OBJ_RECTANGLE_LABEL</a>	Объект "Прямоугольная метка" для создания и оформления пользовательского графического интерфейса.	<a href="#">Типы объектов</a>
<a href="#">OBJ_REGRESSION</a>	Канал на линейной регрессии	<a href="#">Типы объектов</a>
<a href="#">OBJ_STDDEVCCHANNEL</a>	Канал стандартного отклонения	<a href="#">Типы объектов</a>
<a href="#">OBJ_TEXT</a>	Объект "Текст"	<a href="#">Типы объектов</a>
<a href="#">OBJ_TREND</a>	Трендовая линия	<a href="#">Типы объектов</a>
<a href="#">OBJ_TRENDBYANGLE</a>	Трендовая линия по углу	<a href="#">Типы объектов</a>
<a href="#">OBJ_TRIANGLE</a>	Треугольник	<a href="#">Типы объектов</a>
<a href="#">OBJ_VLINE</a>	Вертикальная линия	<a href="#">Типы объектов</a>
OBJPROP_ALIGN	Горизонтальное выравнивание текста в объекте "Поле ввода" (OBJ_EDIT)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_ANCHOR	Положение точки привязки графического объекта	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_ANGLE	Угол. Для объектов с еще не заданным углом, созданных из программы, значение равно <a href="#">EMPTY_VALUE</a>	<a href="#">ObjectSetDouble</a> , <a href="#">ObjectGetDouble</a>
OBJPROP_ARROWCODE	Код стрелки для объекта "Стрелка"	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_BACK	Объект на заднем плане	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>

OBJPROP_BGCOLOR	Цвет фона для OBJ_EDIT, OBJ_BUTTON, OBJ_RECTANGLE_LABEL	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_BMPFILE	Имя BMP-файла для объекта "Графическая метка". Смотри также <a href="#">Ресурсы</a>	<a href="#">ObjectSetString</a> , <a href="#">ObjectGetString</a>
OBJPROP_BORDER_COLOR	Цвет рамки для объекта OBJ_EDIT и OBJ_BUTTON	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_BORDER_TYPE	Тип рамки для объекта "Прямоугольная рамка"	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_CHART_ID	Идентификатор объекта "График" ( <a href="#">OBJ_CHART</a> ). Позволяет работать со свойствами этого объекта как с обычным графиком с помощью функций из раздела <a href="#">Операции с графиками</a> , но есть некоторые <a href="#">исключения</a> .	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_CHART_SCALE	Масштаб для объекта "График"	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_COLOR	Цвет	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_CORNER	Угол графика для привязки графического объекта	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_CREATETIME	Время создания объекта	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_DATE_SCALE	Признак отображения шкалы времени для объекта "График"	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_DEGREE	Уровень волновой разметки Эллиота	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_DEVIATION	Отклонение для канала стандартного отклонения	<a href="#">ObjectSetDouble</a> , <a href="#">ObjectGetDouble</a>
OBJPROP_DIRECTION	Тренд объекта Ганна	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_DRAWLINES	Отображение линий для волновой разметки Эллиота	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_ELLIPSE	Отображение полного эллипса для объекта "Дуги Фибоначчи" ( <a href="#">OBJ_FIBOARC</a> )	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>

OBJPROP_FILL	Заливка объекта цветом (для OBJ_RECTANGLE, OBJ_TRIANGLE, OBJ_ELLIPSE, OBJ_CHANNEL, OBJ_STDDEVCHANNEL, OBJ_REGRESSION)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_FONT	Шрифт	<a href="#">ObjectSetString</a> , <a href="#">ObjectGetString</a>
OBJPROP_FONTSIZE	Размер шрифта	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_HIDDEN	Запрет на показ имени графического объекта в списке объектов из меню терминала "Графики" - "Объекты" - "Список объектов". Значение true позволяет скрыть ненужный для пользователя объект из списка. По умолчанию true устанавливается для объектов, которые отображают события календаря, историю торговли, а также для <a href="#">созданных из MQL5-программы</a> . Для того чтобы увидеть такие <a href="#">графические объекты</a> и получить доступ к их свойствам, нужно нажать кнопку "Все" в окне "Список объектов".	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_LEVELCOLOR	Цвет линии-уровня	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_LEVELS	Количество уровней	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_LEVELSTYLE	Стиль линии-уровня	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_LEVELTEXT	Описание уровня	<a href="#">ObjectSetString</a> , <a href="#">ObjectGetString</a>
OBJPROP_LEVELVALUE	Значение уровня	<a href="#">ObjectSetDouble</a> , <a href="#">ObjectGetDouble</a>
OBJPROP_LEVELWIDTH	Толщина линии-уровня	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>

OBJPROP_NAME	Имя объекта	<a href="#">ObjectSetString</a> , <a href="#">ObjectGetString</a>
OBJPROP_PERIOD	Период для объекта "График"	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_PRICE	Координата цены	<a href="#">ObjectSetDouble</a> , <a href="#">ObjectGetDouble</a>
OBJPROP_PRICE_SCALE	Признак отображения ценовой шкалы для объекта "График"	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_RAY	Вертикальная линия продолжается на все окна графика	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_RAY_LEFT	Луч продолжается влево	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_RAY_RIGHT	Луч продолжается вправо	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_READONLY	Возможность редактирования текста в объекте Edit	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_SCALE	Масштаб (свойство объектов Ганна и объекта "Дуги Фибоначчи")	<a href="#">ObjectSetDouble</a> , <a href="#">ObjectGetDouble</a>
OBJPROP_SELECTABLE	Доступность объекта	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_SELECTED	Выделенность объекта	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_STATE	Состояние кнопки (Нажата/Отжата)	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_STYLE	Стиль	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_SYMBOL	Символ для объекта "График"	<a href="#">ObjectSetString</a> , <a href="#">ObjectGetString</a>
OBJPROP_TEXT	Описание объекта (текст, содержащийся в объекте)	<a href="#">ObjectSetString</a> , <a href="#">ObjectGetString</a>
OBJPROP_TIME	Координата времени	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_TIMEFRAMES	Видимость объекта на таймфреймах	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_TOOLTIP	Текст всплывающей подсказки. Если свойство не задано, то показывается	<a href="#">ObjectSetString</a> , <a href="#">ObjectGetString</a>

	подсказка, автоматически сформированная терминалом. Можно отключить показ подсказки, установив для нее значение "\n" (перевод строки)	
OBJPROP_TYPE	Тип объекта	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_WIDTH	Толщина линии	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_XDISTANCE	Дистанция в пикселях по оси X от угла привязки (см. <a href="#">примечание</a> )	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_XOFFSET	X-координата левого верхнего угла <a href="#">прямоугольной области видимости</a> в графических объектах "Графическая метка" и "Рисунок" (OBJ_BITMAP_LABEL и OBJ_BITMAP). Значение задается в пикселях относительного верхнего левого угла исходного изображения.	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_XSIZE	Ширина объекта по оси X в пикселях. Задается для объектов OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL..	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_YDISTANCE	Дистанция в пикселях по оси Y от угла привязки (см. <a href="#">примечание</a> )	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_YOFFSET	Y-координата левого верхнего угла <a href="#">прямоугольной области видимости</a> в графических объектах "Графическая метка" и "Рисунок" (OBJ_BITMAP_LABEL и OBJ_BITMAP). Значение задается в пикселях относительного верхнего левого угла исходного изображения.	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>

OBJPROP_YSIZE	Высота объекта по оси Y в пикселях. Задается для объектов OBJ_LABEL (read only), OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL.	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
OBJPROP_ZORDER	Приоритет графического объекта на получение события нажатия мышки на графике ( <a href="#">CHARTEVENT_CLICK</a> ). По умолчанию при создании значение выставляется равным нулю, но при необходимости можно повысить приоритет. При наложении объектов друг на друга событие CHARTEVENT_CLICK получит только один объект, чей приоритет выше остальных.	<a href="#">ObjectSetInteger</a> , <a href="#">ObjectGetInteger</a>
ORDER_COMMENT	Комментарий	<a href="#">OrderGetString</a> , <a href="#">HistoryOrderGetString</a>
ORDER_FILLING_FOK	Данная политика исполнения означает, что ордер может быть выполнен исключительно в указанном объеме. Если на рынке в данный момент не присутствует достаточного объема финансового инструмента, то ордер не будет выполнен. Необходимый объем может быть составлен из нескольких предложений, доступных в данный момент на рынке.	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_FILLING_IOC	Означает согласие совершить сделку по максимально доступному на рынке объему в пределах указанного в ордере. В случае невозможности полного исполнения ордер будет выполнен на доступный	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>

	объем, а неисполненный объем ордера будет отменен.	
ORDER_FILLING_RETURN	<p>Данный режим используется для рыночных (ORDER_TYPE_BUY и ORDER_TYPE_SELL), лимитных и стоп-лимитных ордеров (ORDER_TYPE_BUY_LIMIT, ORDER_TYPE_SELL_LIMIT, ORDER_TYPE_BUY_STOP_LIMIT и ORDER_TYPE_SELL_STOP_LIMIT) и только в <a href="#">режимах</a> "Исполнение по рынку" и "Биржевое исполнение". В случае частичного исполнения рыночный или лимитный ордер с остаточным объемом не снимается, а продолжает действовать.</p> <p>Для ордеров ORDER_TYPE_BUY_STOP_LIMIT и ORDER_TYPE_SELL_STOP_LIMIT при активации будет создан соответствующий лимитный ордер ORDER_TYPE_BUY_LIMIT/ORDER_TYPE_SELL_LIMIT с типом исполнения ORDER_FILLING_RETURN.</p>	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_MAGIC	Идентификатор эксперта выставившего ордер (предназначен для того, чтобы каждый эксперт выставлял свой собственный уникальный номер)	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_POSITION_ID	<a href="#">Идентификатор позиции</a> , который ставится на ордере при его исполнении. Каждый исполненный ордер порождает <a href="#"> сделку</a> , которая открывает новую или изменяет уже существующую <a href="#">позицию</a> . Идентификатор этой позиции и	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>

	устанавливается исполненному ордеру в этот момент.	
ORDER_PRICE_CURRENT	Текущая цена по символу ордера	<a href="#">OrderGetDouble</a> , <a href="#">HistoryOrderGetDouble</a>
ORDER_PRICE_OPEN	Цена, указанная в ордере	<a href="#">OrderGetDouble</a> , <a href="#">HistoryOrderGetDouble</a>
ORDER_PRICE_STOPLIMIT	Цена постановки Limit ордера при срабатывании StopLimit ордера	<a href="#">OrderGetDouble</a> , <a href="#">HistoryOrderGetDouble</a>
ORDER_SL	Уровень Stop Loss	<a href="#">OrderGetDouble</a> , <a href="#">HistoryOrderGetDouble</a>
ORDER_STATE	Статус ордера	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_STATE_CANCELED	Ордер снят клиентом	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_STATE_EXPIRED	Ордер снят по истечении срока его действия	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_STATE_FILLED	Ордер выполнен полностью	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_STATE_PARTIAL	Ордер выполнен частично	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_STATE_PLACED	Ордер принят	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_STATE_REJECTED	Ордер отклонен	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_STATE_REQUEST_ADD	Ордер в состоянии регистрации (выставление в торговую систему)	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_STATE_REQUEST_CANCEL	Ордер в состоянии удаления (удаление из торговой системы)	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_STATE_REQUEST_MODIFY	Ордер в состоянии модификации (изменение его параметров)	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_STATE_STARTED	Ордер проверен на корректность, но еще не принят брокером	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_SYMBOL	Символ, по которому выставлен ордер	<a href="#">OrderGetString</a> , <a href="#">HistoryOrderGetString</a>

ORDER_TIME_DAY	Ордер будет действовать только в течение текущего торгового дня	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TIME_DONE	Время исполнения или снятия ордера	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TIME_DONE_MSC	Время исполнения/снятия ордера в миллисекундах с 01.01.1970	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TIME_EXPIRATION	Время истечения ордера	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TIME_GTC	Ордер будет находиться в очереди до тех пор, пока не будет снят	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TIME_SETUP	Время постановки ордера	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TIME_SETUP_MSC	Время установки ордера на исполнение в миллисекундах с 01.01.1970	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TIME_SPECIFIED	Ордер будет действовать до даты истечения	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TIME_SPECIFIED_DAY	Ордер будет действовать до 23:59:59 указанного дня. Если это время не попадает на торговую сессию, истечение наступит в ближайшее торговое время.	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TP	Уровень Take Profit	<a href="#">OrderGetDouble</a> , <a href="#">HistoryOrderGetDouble</a>
ORDER_TYPE	Тип ордера	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TYPE_BUY	Рыночный ордер на покупку	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TYPE_BUY_LIMIT	Отложенный ордер Buy Limit	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TYPE_BUY_STOP	Отложенный ордер Buy Stop	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TYPE_BUY_STOP_LIMIT	По достижении цены ордера выставляется отложенный ордер Buy Limit по цене StopLimit	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>

ORDER_TYPE_FILLING	Тип исполнения по остатку	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TYPE_SELL	Рыночный ордер на продажу	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TYPE_SELL_LIMIT	Отложенный ордер Sell Limit	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TYPE_SELL_STOP	Отложенный ордер Sell Stop	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TYPE_SELL_STOP_LIMIT	По достижении цены ордера выставляется отложенный ордер Sell Limit по цене StopLimit	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_TYPE_TIME	Время жизни ордера	<a href="#">OrderGetInteger</a> , <a href="#">HistoryOrderGetInteger</a>
ORDER_VOLUME_CURRENT	Невыполненный объем	<a href="#">OrderGetDouble</a> , <a href="#">HistoryOrderGetDouble</a>
ORDER_VOLUME_INITIAL	Первоначальный объем при постановке ордера	<a href="#">OrderGetDouble</a> , <a href="#">HistoryOrderGetDouble</a>
PERIOD_CURRENT	Текущий период	<a href="#">Периоды графиков</a>
PERIOD_D1	1 день	<a href="#">Периоды графиков</a>
PERIOD_H1	1 час	<a href="#">Периоды графиков</a>
PERIOD_H12	12 часов	<a href="#">Периоды графиков</a>
PERIOD_H2	2 часа	<a href="#">Периоды графиков</a>
PERIOD_H3	3 часа	<a href="#">Периоды графиков</a>
PERIOD_H4	4 часа	<a href="#">Периоды графиков</a>
PERIOD_H6	6 часов	<a href="#">Периоды графиков</a>
PERIOD_H8	8 часов	<a href="#">Периоды графиков</a>
PERIOD_M1	1 минута	<a href="#">Периоды графиков</a>
PERIOD_M10	10 минут	<a href="#">Периоды графиков</a>
PERIOD_M12	12 минут	<a href="#">Периоды графиков</a>
PERIOD_M15	15 минут	<a href="#">Периоды графиков</a>
PERIOD_M2	2 минуты	<a href="#">Периоды графиков</a>
PERIOD_M20	20 минут	<a href="#">Периоды графиков</a>
PERIOD_M3	3 минуты	<a href="#">Периоды графиков</a>
PERIOD_M30	30 минут	<a href="#">Периоды графиков</a>

PERIOD_M4	4 минуты	<a href="#">Периоды графиков</a>
PERIOD_M5	5 минут	<a href="#">Периоды графиков</a>
PERIOD_M6	6 минут	<a href="#">Периоды графиков</a>
PERIOD_MN1	1 месяц	<a href="#">Периоды графиков</a>
PERIOD_W1	1 неделя	<a href="#">Периоды графиков</a>
PLOT_ARROW	Код стрелки для стиля DRAW_ARROW	<a href="#">PlotIndexSetInteger</a> , <a href="#">PlotIndexGetInteger</a>
PLOT_ARROW_SHIFT	Смещение стрелок по вертикали для стиля DRAW_ARROW	<a href="#">PlotIndexSetInteger</a> , <a href="#">PlotIndexGetInteger</a>
PLOT_COLOR_INDEXES	Количество цветов	<a href="#">PlotIndexSetInteger</a> , <a href="#">PlotIndexGetInteger</a>
PLOT_DRAW_BEGIN	Количество начальных баров без отрисовки и значений в DataWindow	<a href="#">PlotIndexSetInteger</a> , <a href="#">PlotIndexGetInteger</a>
PLOT_DRAW_TYPE	Тип графического построения	<a href="#">PlotIndexSetInteger</a> , <a href="#">PlotIndexGetInteger</a>
PLOT_EMPTY_VALUE	Пустое значение для построения, для которого нет отрисовки	<a href="#">PlotIndexSetDouble</a>
PLOT_LABEL	Имя индикаторной графической серии для отображения в окне DataWindow. Для сложных графических стилей, требующих для отображения несколько индикаторных буферов, имена для каждого буфера можно задать с использованием ";" в качестве разделителя. Пример кода приведен в <a href="#">DRAW_CANDLES</a>	<a href="#">PlotIndexSetString</a>
PLOT_LINE_COLOR	Индекс буфера, содержащего цвет отрисовки	<a href="#">PlotIndexSetInteger</a> , <a href="#">PlotIndexGetInteger</a>
PLOT_LINE_STYLE	Стиль линии отрисовки	<a href="#">PlotIndexSetInteger</a> , <a href="#">PlotIndexGetInteger</a>
PLOT_LINE_WIDTH	Толщина линии отрисовки	<a href="#">PlotIndexSetInteger</a> , <a href="#">PlotIndexGetInteger</a>
PLOT_SHIFT	Сдвиг графического построения индикатора по	<a href="#">PlotIndexSetInteger</a> , <a href="#">PlotIndexGetInteger</a>

	оси времени в барах	
PLOT_SHOW_DATA	Признак отображения значений построения в окне DataWindow	<a href="#">PlotIndexSetInteger</a> , <a href="#">PlotIndexGetInteger</a>
PLUSDI_LINE	Линия +DI	<a href="#">Линии индикаторов</a>
POINTER_AUTOMATIC	Указатель любого объекта, созданного автоматически (без использования new())	<a href="#">CheckPointer</a>
POINTER_DYNAMIC	Указатель объекта, созданного оператором <code>new</code>	<a href="#">CheckPointer</a>
POINTER_INVALID	Некорректный указатель	<a href="#">CheckPointer</a>
POSITION_COMMENT	Комментарий к позиции	<a href="#">PositionGetString</a>
POSITION_COMMISSION	Комиссия	<a href="#">PositionGetDouble</a>
POSITION_IDENTIFIER	Идентификатор позиции - это уникальное число, которое присваивается каждой вновь открытой позиции и не изменяется в течение всей ее жизни. Переворот позиции не изменяет идентификатора позиции.	<a href="#">PositionGetInteger</a>
POSITION_MAGIC	Magic number для позиции (смотри <a href="#">ORDER_MAGIC</a> )	<a href="#">PositionGetInteger</a>
POSITION_PRICE_CURRENT	Текущая цена по символу	<a href="#">PositionGetDouble</a>
POSITION_PRICE_OPEN	Цена позиции	<a href="#">PositionGetDouble</a>
POSITION_PROFIT	Текущая прибыль	<a href="#">PositionGetDouble</a>
POSITION_SL	Уровень Stop Loss для открытой позиции	<a href="#">PositionGetDouble</a>
POSITION_SWAP	Накопленный своп	<a href="#">PositionGetDouble</a>
POSITION_SYMBOL	Символ, по которому открыта позиция	<a href="#">PositionGetString</a>
POSITION_TIME	Время открытия позиции	<a href="#">PositionGetInteger</a>
POSITION_TIME_MSC	Время открытия позиции в миллисекундах с 01.01.1970	<a href="#">PositionGetInteger</a>
POSITION_TIME_UPDATE	Время изменения позиции в секундах с 01.01.1970	<a href="#">PositionGetInteger</a>
POSITION_TIME_UPDATE_MSC	Время изменения позиции в миллисекундах с 01.01.1970	<a href="#">PositionGetInteger</a>

POSITION_TP	Уровень Take Profit для открытой позиции	<a href="#">PositionGetDouble</a>
POSITION_TYPE	Тип позиции	<a href="#">PositionGetInteger</a>
POSITION_TYPE_BUY	Покупка	<a href="#">PositionGetInteger</a>
POSITION_TYPE_SELL	Продажа	<a href="#">PositionGetInteger</a>
POSITION_VOLUME	Объем позиции	<a href="#">PositionGetDouble</a>
PRICE_CLOSE	Цена закрытия	<a href="#">Ценовые константы</a>
PRICE_HIGH	Максимальная за период цена	<a href="#">Ценовые константы</a>
PRICE_LOW	Минимальная за период цена	<a href="#">Ценовые константы</a>
PRICE_MEDIAN	Медианная цена, $(high+low)/2$	<a href="#">Ценовые константы</a>
PRICE_OPEN	Цена открытия	<a href="#">Ценовые константы</a>
PRICE_TYPICAL	Типичная цена, $(high+low+close)/3$	<a href="#">Ценовые константы</a>
PRICE_WEIGHTED	Средневзвешенная цена, $(high+low+close+close)/4$	<a href="#">Ценовые константы</a>
PROGRAM_EXPERT	Эксперт	<a href="#">MQLInfoInteger</a>
PROGRAM_INDICATOR	Индикатор	<a href="#">MQLInfoInteger</a>
PROGRAM_SCRIPT	Скрипт	<a href="#">MQLInfoInteger</a>
REASON_ACCOUNT	Активирован другой счет либо произошло переподключение к торговому серверу вследствие изменения настроек счета	<a href="#">UninitializeReason</a> , <a href="#">OnDeinit</a>
REASON_CHARTCHANGE	Символ или период графика был изменен	<a href="#">UninitializeReason</a> , <a href="#">OnDeinit</a>
REASON_CHARTCLOSE	График закрыт	<a href="#">UninitializeReason</a> , <a href="#">OnDeinit</a>
REASON_CLOSE	Терминал был закрыт	<a href="#">UninitializeReason</a> , <a href="#">OnDeinit</a>
REASON_INITFAILED	Признак того, что обработчик <a href="#">OnInit()</a> вернул ненулевое значение	<a href="#">UninitializeReason</a> , <a href="#">OnDeinit</a>
REASON_PARAMETERS	Входные параметры были изменены пользователем	<a href="#">UninitializeReason</a> , <a href="#">OnDeinit</a>
REASON_PROGRAM	Эксперт прекратил свою работу, вызвав функцию	<a href="#">UninitializeReason</a> , <a href="#">OnDeinit</a>

	<a href="#">ExpertRemove()</a>	
REASON_RECOMPILE	Программа перекомпилирована	<a href="#">UninitializeReason</a> , <a href="#">OnDeinit</a>
REASON_REMOVE	Программа удалена с графика	<a href="#">UninitializeReason</a> , <a href="#">OnDeinit</a>
REASON_TEMPLATE	Применен другой шаблон графика	<a href="#">UninitializeReason</a> , <a href="#">OnDeinit</a>
SATURDAY	Суббота	<a href="#">SymbolInfoInteger</a> , <a href="#">SymbolInfoSessionQuote</a> , <a href="#">SymbolInfoSessionTrade</a>
SEEK_CUR	Текущая позиция файлового указателя	<a href="#">FileSeek</a>
SEEK_END	Конец файла	<a href="#">FileSeek</a>
SEEK_SET	Начало файла	<a href="#">FileSeek</a>
SENKOUSPANA_LINE	Линия Senkou Span A	<a href="#">Линии индикаторов</a>
SENKOUSPANB_LINE	Линия Senkou Span B	<a href="#">Линии индикаторов</a>
SERIES_BARS_COUNT	Количество баров по символу-периоду на данный момент	<a href="#">SeriesInfoInteger</a>
SERIES_FIRSTDATE	Самая первая дата по символу-периоду на данный момент	<a href="#">SeriesInfoInteger</a>
SERIES_LASTBAR_DATE	Время открытия последнего бара по символу-периоду	<a href="#">SeriesInfoInteger</a>
SERIES_SERVER_FIRSTDATE	Самая первая дата в истории по символу на сервере независимо от периода	<a href="#">SeriesInfoInteger</a>
SERIES_SYNCHRONIZED	Признак синхронизированности данных по символу/периоду на данный момент	<a href="#">SeriesInfoInteger</a>
SERIES_TERMINAL_FIRSTDATE	Самая первая дата в истории по символу в клиентском терминале независимо от периода	<a href="#">SeriesInfoInteger</a>
SHORT_MAX	Максимальное значение, которое может быть представлено типом short	<a href="#">Константы числовых типов</a>
SHORT_MIN	Минимальное значение, которое может быть	<a href="#">Константы числовых типов</a>

	представлено типом short	
SIGNAL_BASE_AUTHOR_LOGIN	Логин автора сигнала	<a href="#">SignalBaseGetString</a>
SIGNAL_BASE_BALANCE	Баланс счета	<a href="#">SignalBaseGetDouble</a>
SIGNAL_BASE_BROKER	Наименование брокера (компании)	<a href="#">SignalBaseGetString</a>
SIGNAL_BASE_BROKER_SERVER	Сервер брокера	<a href="#">SignalBaseGetString</a>
SIGNAL_BASE_CURRENCY	Валюта счета сигнала	<a href="#">SignalBaseGetString</a>
SIGNAL_BASE_DATE_PUBLISHED	Дата публикации сигнала (когда стал доступен для подписки)	<a href="#">SignalBaseGetInteger</a>
SIGNAL_BASE_DATE_STARTED	Дата начала мониторинга сигнала	<a href="#">SignalBaseGetInteger</a>
SIGNAL_BASE_EQUITY	Средства на счете	<a href="#">SignalBaseGetDouble</a>
SIGNAL_BASE_GAIN	Прирост счета в процентах	<a href="#">SignalBaseGetDouble</a>
SIGNAL_BASE_ID	ID сигнала	<a href="#">SignalBaseGetInteger</a>
SIGNAL_BASE_LEVERAGE	Плечо торгового счета	<a href="#">SignalBaseGetInteger</a>
SIGNAL_BASE_MAX_DRAWDOWN	Максимальная просадка	<a href="#">SignalBaseGetDouble</a>
SIGNAL_BASE_NAME	Имя сигнала	<a href="#">SignalBaseGetString</a>
SIGNAL_BASE_PIPS	Результат торговли в пипсах	<a href="#">SignalBaseGetInteger</a>
SIGNAL_BASE_PRICE	Цена подписки на сигнал	<a href="#">SignalBaseGetDouble</a>
SIGNAL_BASE_RATING	Позиция в рейтинге сигналов	<a href="#">SignalBaseGetInteger</a>
SIGNAL_BASE_ROI	Значение ROI (Return on Investment) сигнала в %	<a href="#">SignalBaseGetDouble</a>
SIGNAL_BASE_SUBSCRIBERS	Количество подписчиков	<a href="#">SignalBaseGetInteger</a>
SIGNAL_BASE_TRADE_MODE	Тип счета (0-реальный счет, 1-демо-счет, 2-конкурсный счет)	<a href="#">SignalBaseGetInteger</a>
SIGNAL_BASE_TRADES	Количество трейдов	<a href="#">SignalBaseGetInteger</a>
SIGNAL_INFO_CONFIRMATION_S_DISABLED	Флаг разрешения синхронизации без показа диалога подтверждения	<a href="#">SignalInfoGetInteger</a> , <a href="#">SignalInfoSetInteger</a>
SIGNAL_INFO_COPY_SLTP	Флаг копирования Stop Loss и Take Profit	<a href="#">SignalInfoGetInteger</a> , <a href="#">SignalInfoSetInteger</a>

SIGNAL_INFO_DEPOSIT_PERCENT	Ограничения по депозиту (в %)	<a href="#">SignalInfoGetInteger</a> , <a href="#">SignalInfoSetInteger</a>
SIGNAL_INFO_EQUITY_LIMIT	Процент для конвертации объема сделки	<a href="#">SignalInfoGetDouble</a> , <a href="#">SignalInfoSetDouble</a>
SIGNAL_INFO_ID	id сигнала, r/o	<a href="#">SignalInfoGetInteger</a> , <a href="#">SignalInfoSetInteger</a>
SIGNAL_INFO_NAME	Имя сигнала, r/o	<a href="#">SignalInfoGetString</a>
SIGNAL_INFO_SLIPPAGE	Величина проскальзывания, с которым выставляются рыночные ордера при синхронизации позиций и копировании сделок	<a href="#">SignalInfoGetDouble</a> , <a href="#">SignalInfoSetDouble</a>
SIGNAL_INFO_SUBSCRIPTION_ENABLED	Флаг разрешения на копирование сделок по подписке	<a href="#">SignalInfoGetInteger</a> , <a href="#">SignalInfoSetInteger</a>
SIGNAL_INFO_TERMS_AGREE	Флаг согласия с условиями использования сервиса "Сигналы", r/o	<a href="#">SignalInfoGetInteger</a> , <a href="#">SignalInfoSetInteger</a>
SIGNAL_INFO_VOLUME_PERCENT	Значение ограничения по средствам для сигнала, r/o	<a href="#">SignalInfoGetDouble</a> , <a href="#">SignalInfoSetDouble</a>
SIGNAL_LINE	Сигнальная линия	Линии индикаторов
STAT_BALANCE_DD	Максимальная просадка баланса в деньгах. В процессе торговли баланс может испытать множество просадок, берется наибольшее значение.	<a href="#">TesterStatistics</a>
STAT_BALANCE_DD_RELATIVE	Просадка баланса в деньгах, которая была зафиксирована в момент максимальной просадки баланса в процентах (STAT_BALANCE_DDREL_PERCENT).	<a href="#">TesterStatistics</a>
STAT_BALANCE_DDREL_PERCENT	Максимальная просадка баланса в процентах. В процессе торговли баланс может испытать множество просадок, для каждой фиксируется относительное значение просадки в процентах. Возвращается наибольшее значение	<a href="#">TesterStatistics</a>

STAT_BALANCEDD_PERCENT	Просадка баланса в процентах, которая была зафиксирована в момент максимальной просадки баланса в деньгах (STAT_BALANCE_DD).	<a href="#">TesterStatistics</a>
STAT_BALANCEMIN	Минимальное значение баланса	<a href="#">TesterStatistics</a>
STAT_CONLOSSMAX	Максимальный убыток в последовательности убыточных трейдов. Значение меньше или равно нулю	<a href="#">TesterStatistics</a>
STAT_CONLOSSMAX_TRADES	Количество трейдов, сформировавших STAT_CONLOSSMAX (максимальный убыток в последовательности убыточных трейдов)	<a href="#">TesterStatistics</a>
STAT_CONPROFITMAX	Максимальная прибыль в последовательности прибыльных трейдов. Значение больше или равно нулю	<a href="#">TesterStatistics</a>
STAT_CONPROFITMAX_TRADES	Количество трейдов, сформировавших STAT_CONPROFITMAX (максимальная прибыль в последовательности прибыльных трейдов)	<a href="#">TesterStatistics</a>
STAT_CUSTOM_ONTESTER	Значение рассчитанного пользовательского критерия оптимизации, возвращаемого функцией <a href="#">OnTester()</a>	<a href="#">TesterStatistics</a>
STAT DEALS	Количество совершенных сделок	<a href="#">TesterStatistics</a>
STAT_EQUITY_DD	Максимальная просадка средств в деньгах. В процессе торговли средства могут испытать множество просадок, берется наибольшее значение.	<a href="#">TesterStatistics</a>
STAT_EQUITY_DD_RELATIVE	Просадка средств в деньгах, которая была зафиксирована в момент максимальной	<a href="#">TesterStatistics</a>

	просадки средств в процентах (STAT_EQUITY_DDREL_PERCENT).	
STAT_EQUITY_DDREL_PERCENT	Максимальная просадка средств в процентах. В процессе торговли средства могут испытать множество просадок, для каждой фиксируется относительное значение просадки в процентах. Возвращается наибольшее значение	<a href="#">TesterStatistics</a>
STAT_EQUITYDD_PERCENT	Просадка средств в процентах, которая была зафиксирована в момент максимальной просадки средств в деньгах (STAT_EQUITY_DD).	<a href="#">TesterStatistics</a>
STAT_EQUITYMIN	Минимальное значение собственных средств	<a href="#">TesterStatistics</a>
STAT_EXPECTED_PAYOFF	Математическое ожидание выигрыша	<a href="#">TesterStatistics</a>
STAT_GROSS_LOSS	Общий убыток, сумма всех убыточных (отрицательных) трейдов. Значение меньше или равно нулю	<a href="#">TesterStatistics</a>
STAT_GROSS_PROFIT	Общая прибыль, сумма всех прибыльных (положительных) трейдов. Значение больше или равно нулю	<a href="#">TesterStatistics</a>
STAT_INITIAL_DEPOSIT	Значение начального депозита	<a href="#">TesterStatistics</a>
STAT_LONG_TRADES	Длинные трейды	<a href="#">TesterStatistics</a>
STAT_LOSS_TRADES	Убыточные трейды	<a href="#">TesterStatistics</a>
STAT_LOSSTRADES_AVGCON	Средняя длина убыточной серии трейдов	<a href="#">TesterStatistics</a>
STAT_MAX_CONLOSS_TRADES	Количество трейдов в самой длинной серии убыточных трейдов STAT_MAX_CONLOSSES	<a href="#">TesterStatistics</a>

STAT_MAX_CONLOSSES	Общий убыток в самой длинной серии убыточных трейдов	<a href="#">TesterStatistics</a>
STAT_MAX_CONPROFIT_TRADES	Количество трейдов в самой длинной серии прибыльных трейдов STAT_MAX_CONWINS	<a href="#">TesterStatistics</a>
STAT_MAX_CONWINS	Общая прибыль в самой длинной серии прибыльных трейдов	<a href="#">TesterStatistics</a>
STAT_MAX_LOSSTRADE	Максимальный убыток - наименьшее значение среди всех убыточных трейдов. Значение меньше или равно нулю	<a href="#">TesterStatistics</a>
STAT_MAX_PROFITTRADE	Максимальная прибыль - наибольшее значение среди всех прибыльных трейдов. Значение больше или равно нулю	<a href="#">TesterStatistics</a>
STAT_MIN_MARGINLEVEL	Минимальное достигнутое значение уровня маржи	<a href="#">TesterStatistics</a>
STAT_PROFIT	Чистая прибыль по окончании тестирования, сумма STAT_GROSS_PROFIT и STAT_GROSS_LOSS (STAT_GROSS_LOSS всегда меньше или равно нулю)	<a href="#">TesterStatistics</a>
STAT_PROFIT_FACTOR	Прибыльность - отношение STAT_GROSS_PROFIT/STAT_GROSS_LOSS. Если STAT_GROSS_LOSS=0, то прибыльность принимает значение <a href="#">DBL_MAX</a>	<a href="#">TesterStatistics</a>
STAT_PROFIT_LONGTRADES	Длинные прибыльные трейды	<a href="#">TesterStatistics</a>
STAT_PROFIT_SHORTTRADES	Короткие прибыльные трейды	<a href="#">TesterStatistics</a>
STAT_PROFIT_TRADES	Прибыльные трейды	<a href="#">TesterStatistics</a>
STAT_PROFITTRADES_AVGCON	Средняя длина прибыльной серии трейдов	<a href="#">TesterStatistics</a>
STAT_RECOVERY_FACTOR	Фактор восстановления - отношение STAT_PROFIT/STAT_BALANCE_DD	<a href="#">TesterStatistics</a>

STAT_SHARPE_RATIO	Коэффициент Шарпа	<a href="#">TesterStatistics</a>
STAT_SHORT_TRADES	Короткие трейды	<a href="#">TesterStatistics</a>
STAT_TRADES	Количество трейдов	<a href="#">TesterStatistics</a>
STAT_WITHDRAWAL	Количество выведенных со счета средств	<a href="#">TesterStatistics</a>
STO_CLOSECLOSE	Построение по ценам Close/Close	<a href="#">Ценовые константы</a>
STO_LWHIGH	Построение по ценам Low/High	<a href="#">Ценовые константы</a>
STYLE_DASH	Прерывистая линия	<a href="#">Стили рисования</a>
STYLE_DASHDOT	Штрих-пунктирная линия	<a href="#">Стили рисования</a>
STYLE_DASHDOTDOT	Штрих - две точки	<a href="#">Стили рисования</a>
STYLE_DOT	Пунктирная линия	<a href="#">Стили рисования</a>
STYLE_SOLID	Сплошная линия	<a href="#">Стили рисования</a>
SUNDAY	Воскресенье	<a href="#">SymbolInfoInteger</a> , <a href="#">SymbolInfoSessionQuote</a> , <a href="#">SymbolInfoSessionTrade</a>
SYMBOL_ASK	Ask - лучшее предложение на покупку	<a href="#">SymbolInfoDouble</a>
SYMBOL_ASKHIGH	Максимальный Ask за день	<a href="#">SymbolInfoDouble</a>
SYMBOL_ASKLOW	Минимальный Ask за день	<a href="#">SymbolInfoDouble</a>
SYMBOL_BANK	Источник текущей котировки	<a href="#">SymbolInfoString</a>
SYMBOL_BASIS	Имя базового актива для производного инструмента	<a href="#">SymbolInfoString</a>
SYMBOL_BID	Bid - лучшее предложение на продажу	<a href="#">SymbolInfoDouble</a>
SYMBOL_BIDHIGH	Максимальный Bid за день	<a href="#">SymbolInfoDouble</a>
SYMBOL_BIDLOW	Минимальный Bid за день	<a href="#">SymbolInfoDouble</a>
SYMBOL_CALC_MODE_CFD	CFD mode - расчет залога и прибыли для CFD	<a href="#">SymbolInfoInteger</a>
SYMBOL_CALC_MODE_CFDINDEX	CFD index mode - расчет залога и прибыли для CFD на индексы	<a href="#">SymbolInfoInteger</a>
SYMBOL_CALC_MODE_CFDLEVERAGE	CFD Leverage mode - расчет залога и прибыли для CFD при торговле с плечом	<a href="#">SymbolInfoInteger</a>

SYMBOL_CALC_MODE_EXCH_FUTURES	Futures mode - расчет залога и прибыли для торговли фьючерсными контрактами на бирже	<a href="#">SymbolInfoInteger</a>
SYMBOL_CALC_MODE_EXCH_FUTURES_FORTS	FORTS Futures mode - расчет залога и прибыли для торговли фьючерсными контрактами на FORTS.	<a href="#">SymbolInfoInteger</a>
SYMBOL_CALC_MODE_EXCH_STOCKS	Exchange mode - расчет залога и прибыли для торговли ценными бумагами на бирже	<a href="#">SymbolInfoInteger</a>
SYMBOL_CALC_MODE_FOREX	Forex mode - расчет прибыли и маржи для Форекс	<a href="#">SymbolInfoInteger</a>
SYMBOL_CALC_MODE_FUTURES	Futures mode - расчет залога и прибыли для фьючерсов	<a href="#">SymbolInfoInteger</a>
SYMBOL_CALC_MODE_SERV_COLLATERAL	Collateral mode - инструмент используется в качестве неторгуемого актива на торговом счете.	<a href="#">SymbolInfoInteger</a>
SYMBOL_CURRENCY_BASE	Базовая валюта инструмента	<a href="#">SymbolInfoString</a>
SYMBOL_CURRENCY_MARGIN	Валюта в которой вычисляются залоговые средства	<a href="#">SymbolInfoString</a>
SYMBOL_CURRENCY_PROFIT	Валюта прибыли	<a href="#">SymbolInfoString</a>
SYMBOL_DESCRIPTION	Строковое описание символа	<a href="#">SymbolInfoString</a>
SYMBOL_DIGITS	Количество знаков после запятой	<a href="#">SymbolInfoInteger</a>
SYMBOL_EXPIRATION_DAY	Ордер действителен до конца дня	<a href="#">SymbolInfoInteger</a>
SYMBOL_EXPIRATION_GTC	Ордер действителен неограниченно по времени до явной его отмены	<a href="#">SymbolInfoInteger</a>
SYMBOL_EXPIRATION_MODE	Флаги разрешенных <a href="#">режимов истечения</a> ордера	<a href="#">SymbolInfoInteger</a>
SYMBOL_EXPIRATION_SPECIFIED	Срок истечения указывается в ордере	<a href="#">SymbolInfoInteger</a>
SYMBOL_EXPIRATION_SPECIFIED_DAY	День истечения указывается в ордере	<a href="#">SymbolInfoInteger</a>

SYMBOL_EXPIRATION_TIME	Дата окончания торгов по инструменту (обычно используется для фьючерсов)	<a href="#">SymbolInfoInteger</a>
SYMBOL_FILLING_FOK	Данная политика исполнения означает, что ордер может быть выполнен исключительно в указанном объеме. Если на рынке в данный момент не присутствует достаточного объема финансового инструмента, то ордер не будет исполнен. Необходимый объем может быть составлен из нескольких предложений, доступных в данный момент на рынке.	<a href="#">SymbolInfoInteger</a>
SYMBOL_FILLING_IOC	В данном случае трейдер соглашается совершить сделку по максимально доступному на рынке объему в пределах указанного в ордере. В случае невозможности полного исполнения ордер будет исполнен на доступный объем, а неисполненный объем ордера будет отменен. Возможность использования IOC ордеров определяется на торговом сервере.	<a href="#">SymbolInfoInteger</a>
SYMBOL_FILLING_MODE	Флаги разрешенных <a href="#">режимов заливки</a> ордера	<a href="#">SymbolInfoInteger</a>
SYMBOL_ISIN	Имя торгового символа в системе международных идентификационных кодов ценных бумаг – ISIN (International Securities Identification Number). Международный идентификационный код ценной бумаги – это 12-разрядный буквенно-цифровой код, однозначно идентифицирующий ценную бумагу. Наличие данного свойства символа	<a href="#">SymbolInfoString</a>

	определяется на стороне торгового сервера.	
SYMBOL_LAST	Цена, по которой совершена последняя сделка	<a href="#">SymbolInfoDouble</a>
SYMBOL_LASTHIGH	Максимальный Last за день	<a href="#">SymbolInfoDouble</a>
SYMBOL_LASTLOW	Минимальный Last за день	<a href="#">SymbolInfoDouble</a>
SYMBOL_MARGIN_INITIAL	Начальная (инициирующая) маржа обозначает размер необходимых залоговых средств в маржинальной валюте для открытия позиции объемом в один лот. Используется при проверке средств клиента при входе в рынок.	<a href="#">SymbolInfoDouble</a>
SYMBOL_MARGIN_MAINTENANCE	Поддерживающая маржа по инструменту. В случае если задана - указывает размер маржи в маржинальной валюте инструмента, удерживаемой с одного лота. Используется при проверке средств клиента при изменении состояния счета клиента. Если поддерживающая маржа равна 0, то используется начальная маржа.	<a href="#">SymbolInfoDouble</a>
SYMBOL_OPTION_MODE	Тип опциона	<a href="#">SymbolInfoInteger</a>
SYMBOL_OPTION_MODE_EUROPEAN	Европейский тип опциона - может быть погашен только в указанную дату (дата истечения срока, дата исполнения, дата погашения)	<a href="#">SymbolInfoInteger</a>
SYMBOL_OPTION_MODE_AMERICAN	Американский тип опциона - может быть погашен в любой день до истечения срока опциона. Для такого типа задается период, в течение которого покупатель может исполнить данный опцион	<a href="#">SymbolInfoInteger</a>
SYMBOL_OPTION_RIGHT	Право опциона (Call/Put)	<a href="#">SymbolInfoInteger</a>
SYMBOL_OPTION_RIGHT_CALL	Опцион, дающий право купить актив по	<a href="#">SymbolInfoInteger</a>

	фиксированной цене	
SYMBOL_OPTION_RIGHT_PUT	Опцион, дающий право продать актив по фиксированной цене	<a href="#">SymbolInfoInteger</a>
SYMBOL_OPTION_STRIKE	Цена исполнения опциона. Это цена, по которой покупатель опциона может купить (при опционе Call) или продать (при опционе Put) базовый актив, а продавец опциона соответственно обязан продать или купить соответствующее количество базового актива.	<a href="#">SymbolInfoDouble</a>
SYMBOL_ORDER_LIMIT	Разрешены лимитные ордера (Buy Limit и Sell Limit)	<a href="#">SymbolInfoInteger</a>
SYMBOL_ORDER_MARKET	Разрешены рыночные ордера (Buy и Sell)	<a href="#">SymbolInfoInteger</a>
SYMBOL_ORDER_MODE	Флаги разрешенных <a href="#">типов ордера</a>	<a href="#">SymbolInfoInteger</a>
SYMBOL_ORDER_SL	Разрешена установка Stop Loss	<a href="#">SymbolInfoInteger</a>
SYMBOL_ORDER_STOP	Разрешены стоп-ордера (Buy Stop и Sell Stop)	<a href="#">SymbolInfoInteger</a>
SYMBOL_ORDER_STOP_LIMIT	Разрешены стоп-лимит ордера (Buy Stop Limit и Sell Stop Limit)	<a href="#">SymbolInfoInteger</a>
SYMBOL_ORDER_TP	Разрешена установка Take Profit	<a href="#">SymbolInfoInteger</a>
SYMBOL_PATH	Путь в дереве символов	<a href="#">SymbolInfoString</a>
SYMBOL_POINT	Значение одного пункта	<a href="#">SymbolInfoDouble</a>
SYMBOL_SELECT	Признак того, что символ выбран в Market Watch	<a href="#">SymbolInfoInteger</a>
SYMBOL_SESSION_AW	Средневзвешенная цена сессии	<a href="#">SymbolInfoDouble</a>
SYMBOL_SESSION_BUY_ORDERS	Общее число ордеров на покупку в текущий момент	<a href="#">SymbolInfoInteger</a>
SYMBOL_SESSION_BUY_ORDERS_VOLUME	Общий объём ордеров на покупку в текущий момент	<a href="#">SymbolInfoDouble</a>
SYMBOL_SESSION_CLOSE	Цена закрытия сессии	<a href="#">SymbolInfoDouble</a>

SYMBOL_SESSION DEALS	Количество сделок в текущей сессии	<a href="#">SymbolInfoInteger</a>
SYMBOL_SESSION INTEREST	Суммарный объём открытых позиций	<a href="#">SymbolInfoDouble</a>
SYMBOL_SESSION OPEN	Цена открытия сессии	<a href="#">SymbolInfoDouble</a>
SYMBOL_SESSION PRICE LIMIT_MAX	Максимально допустимое значение цены на сессию	<a href="#">SymbolInfoDouble</a>
SYMBOL_SESSION PRICE LIMIT_MIN	Минимально допустимое значение цены на сессию	<a href="#">SymbolInfoDouble</a>
SYMBOL_SESSION PRICE SETTLEMENT	Цена поставки на текущую сессию	<a href="#">SymbolInfoDouble</a>
SYMBOL_SESSION SELL ORDERS	Общее число ордеров на продажу в текущий момент	<a href="#">SymbolInfoInteger</a>
SYMBOL_SESSION SELL ORDERS_VOLUME	Общий объём ордеров на продажу в текущий момент	<a href="#">SymbolInfoDouble</a>
SYMBOL_SESSION TURNOVER	Суммарный оборот в текущую сессию	<a href="#">SymbolInfoDouble</a>
SYMBOL_SESSION VOLUME	Суммарный объём сделок в текущую сессию	<a href="#">SymbolInfoDouble</a>
SYMBOL_SPREAD	Размер спреда в пунктах	<a href="#">SymbolInfoInteger</a>
SYMBOL_SPREAD_FLOAT	Признак плавающего спреда	<a href="#">SymbolInfoInteger</a>
SYMBOL_START TIME	Дата начала торгов по инструменту (обычно используется для фьючерсов)	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP LONG	Значение свопа в покупку	<a href="#">SymbolInfoDouble</a>
SYMBOL_SWAP MODE	Модель расчета свопа	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP MODE_CURRENCY_DEPOSIT	Свопы начисляются в деньгах в валюте депозита клиента	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP MODE_CURRENCY_MARGIN	Свопы начисляются в деньгах в маржинальной валюте символа	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP MODE_CURRENCY_SYMBOL	Свопы начисляются в деньгах в базовой валюте символа	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP MODE_DISABLED	Нет свопов	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP MODE_INTEREST_CURRENT	Свопы начисляются в годовых процентах от цены инструмента на момент	<a href="#">SymbolInfoInteger</a>

	расчета свопа(банковский режим - 360 дней в году)	
SYMBOL_SWAP_MODE_INTEREST_OPEN	Свопы начисляются в годовых процентах от цены открытия позиции по символу (банковский режим - 360 дней в году)	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP_MODE_POINTS	Свопы начисляются в пунктах	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP_MODE_REOPEN_BID	Свопы начисляются переоткрытием позиции. В конце торгового дня позиция принудительно закрывается. На следующий день позиция переоткрывается по текущей цене Bid +/- указанное количество пунктов (в параметрах SYMBOL_SWAP_LONG и SYMBOL_SWAP_SHORT)	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP_MODE_REOPEN_CURRENT	Свопы начисляются переоткрытием позиции. В конце торгового дня позиция принудительно закрывается. На следующий день позиция переоткрывается по цене закрытия +/- указанное количество пунктов (в параметрах SYMBOL_SWAP_LONG и SYMBOL_SWAP_SHORT)	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP_ROLLOVER_DAYS	День недели для начисления тройного свопа	<a href="#">SymbolInfoInteger</a>
SYMBOL_SWAP_SHORT	Значение свопа в продажу	<a href="#">SymbolInfoDouble</a>
SYMBOL_TICKS_BOOKDEPTH	Максимальное количество показываемых заявок в <a href="#">стакане</a> . Для инструментов, не имеющих очереди заявок, значение равно 0	<a href="#">SymbolInfoInteger</a>
SYMBOL_TIME	Время последней котировки	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_CALC_MODE	Способ вычисления стоимости контракта	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_CONTRACT_SIZE	Размер торгового контракта	<a href="#">SymbolInfoDouble</a>

SYMBOL_TRADE_EXECUTION_E_XCHANGE	Биржевое исполнение	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_EXECUTION_I_NSTANT	Торговля по потоковым ценам	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_EXECUTION_MARKET	Исполнение ордеров по рынку	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_EXECUTION_R_EQUEST	Торговля по запросу	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_EXEMODE	Режим заключения сделок	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_FREEZE_LEVEL	Дистанция заморозки торговых операций (в пунктах)	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_MODE	Тип исполнения ордеров	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_MODE_CLOSE_ONLY	Разрешены только операции закрытия позиций	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_MODE_DISABLED	Торговля по символу запрещена	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_MODE_FULL	Нет ограничений на торговые операции	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_MODE_LONG_ONLY	Разрешены только покупки	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_MODE_SHORT_ONLY	Разрешены только продажи	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_STOPS_LEVEL	Минимальный отступ в пунктах от текущей цены закрытия для установки Stop ордеров	<a href="#">SymbolInfoInteger</a>
SYMBOL_TRADE_TICK_SIZE	Минимальное изменение цены	<a href="#">SymbolInfoDouble</a>
SYMBOL_TRADE_TICK_VALUE	Значение SYMBOL_TRADE_TICK_VALUE_PROFIT	<a href="#">SymbolInfoDouble</a>
SYMBOL_TRADE_TICK_VALUE_LOSS	Рассчитанная стоимость тика для убыточной позиции	<a href="#">SymbolInfoDouble</a>
SYMBOL_TRADE_TICK_VALUE_PROFIT	Рассчитанная стоимость тика для прибыльной позиции	<a href="#">SymbolInfoDouble</a>
SYMBOL_VOLUME	Volume - объем в последней сделке	<a href="#">SymbolInfoInteger</a>

SYMBOL_VOLUME_LIMIT	Максимально допустимый для данного символа совокупный объем открытой позиции и отложенных ордеров в одном направлении (покупка или продажа). Например, при ограничении в 5 лотов можно иметь открытую позицию на покупку объемом 5 лотов и выставить отложенный ордер Sell Limit объемом 5 лотов. Но при этом нельзя выставить отложенный ордер Buy Limit (поскольку совокупный объем в одном направлении превысит ограничение) или выставить Sell Limit объемом более 5 лотов.	<a href="#">SymbolInfoDouble</a>
SYMBOL_VOLUME_MAX	Максимальный объем для заключения сделки	<a href="#">SymbolInfoDouble</a>
SYMBOL_VOLUME_MIN	Минимальный объем для заключения сделки	<a href="#">SymbolInfoDouble</a>
SYMBOL_VOLUME_STEP	Минимальный шаг изменения объема для заключения сделки	<a href="#">SymbolInfoDouble</a>
SYMBOL_VOLUMEHIGH	Максимальный Volume за день	<a href="#">SymbolInfoInteger</a>
SYMBOL_VOLUMELOW	Минимальный Volume за день	<a href="#">SymbolInfoInteger</a>
TENKANSEN_LINE	Линия Tenkan-sen	<a href="#">Линии индикаторов</a>
TERMINAL_BUILD	Номер билда запущенного терминала	<a href="#">TerminalInfoInteger</a>
TERMINAL_CODEPAGE	Номер <a href="#">кодовой страницы языка</a> , установленного в клиентском терминале	<a href="#">TerminalInfoInteger</a>
TERMINAL_COMMONDATA_PATH	Общая папка всех клиентских терминалов, установленных на компьютере	<a href="#">TerminalInfoString</a>
TERMINAL_COMMUNITY_ACCOUNT	Флаг наличия авторизационных данных MQL5.community в терминале	<a href="#">TerminalInfoInteger</a>

TERMINAL_COMMUNITY_BALANCE	Баланс пользователя в MQL5.community	<a href="#">TerminalInfoDouble</a>
TERMINAL_COMMUNITY_CONNECTION	Наличие подключения к MQL5.community	<a href="#">TerminalInfoInteger</a>
TERMINAL_COMPANY	Имя компании	<a href="#">TerminalInfoString</a>
TERMINAL_CONNECTED	Наличие подключения к торговому серверу	<a href="#">TerminalInfoInteger</a>
TERMINAL_CPU_CORES	Количество процессоров в системе	<a href="#">TerminalInfoInteger</a>
TERMINAL_DATA_PATH	Папка, в которой хранятся данные терминала	<a href="#">TerminalInfoString</a>
TERMINAL_DISK_SPACE	Объем свободной памяти на диске для папки MQL5\Files терминала (агента), в МБ	<a href="#">TerminalInfoInteger</a>
TERMINAL_DLLS_ALLOWED	Разрешение на использование DLL	<a href="#">TerminalInfoInteger</a>
TERMINAL_EMAIL_ENABLED	Разрешение на отправку писем с использованием SMTP-сервера и логина, указанных в настройках терминала	<a href="#">TerminalInfoInteger</a>
TERMINAL_FTP_ENABLED	Разрешение на отправку отчетов по FTP на указанный сервер для указанного в настройках терминала торгового счета	<a href="#">TerminalInfoInteger</a>
TERMINAL_LANGUAGE	Язык терминала	<a href="#">TerminalInfoString</a>
TERMINAL_MAXBARS	Максимальное количество баров на графике	<a href="#">TerminalInfoInteger</a>
TERMINAL_MEMORY_AVAILABLE	Размер свободной памяти процесса терминала (агента) в МБ	<a href="#">TerminalInfoInteger</a>
TERMINAL_MEMORY_PHYSICAL	Размер физической памяти в системе, в МБ	<a href="#">TerminalInfoInteger</a>
TERMINAL_MEMORY_TOTAL	Размер памяти, доступной процессу терминала (агента), в МБ	<a href="#">TerminalInfoInteger</a>
TERMINAL_MEMORY_USED	Размер памяти, использованной терминалом (агентом), в МБ	<a href="#">TerminalInfoInteger</a>

TERMINAL_MQID	Флаг наличия MetaQuotes ID для отправки <a href="#">Push-уведомлений</a>	<a href="#">TerminalInfoInteger</a>
TERMINAL_NAME	Имя терминала	<a href="#">TerminalInfoString</a>
TERMINAL_NOTIFICATIONS_ENABLED	Разрешение на отправку уведомлений на смартфон	<a href="#">TerminalInfoInteger</a>
TERMINAL_OPENCL_SUPPORT	Версия поддерживаемой OpenCL в виде 0x00010002 = 1.2. "0" означает, что OpenCL не поддерживается	<a href="#">TerminalInfoInteger</a>
TERMINAL_PATH	Папка, из которой запущен терминал	<a href="#">TerminalInfoString</a>
TERMINAL_PING_LAST	Последнее известное значение пинга до торгового сервера в микросекундах. В одной секунде миллион микросекунд	<a href="#">TerminalInfoInteger</a>
TERMINAL_SCREEN_DPI	Разрешающая способность вывода информации на экран	<a href="#">TerminalInfoInteger</a>
TERMINAL_TRADE_ALLOWED	<a href="#">Разрешение на торговлю</a>	<a href="#">TerminalInfoInteger</a>
TERMINAL_X64	Признак "64 битный терминал"	<a href="#">TerminalInfoInteger</a>
THURSDAY	Четверг	<a href="#">SymbolInfoInteger</a> , <a href="#">SymbolInfoSessionQuote</a> , <a href="#">SymbolInfoSessionTrade</a>
TRADE_ACTION_DEAL	Установить торговый ордер на немедленное совершение сделки с указанными параметрами (поставить рыночный ордер)	<a href="#">MqlTradeRequest</a>
TRADE_ACTION MODIFY	Изменить параметры ранее установленного торгового ордера	<a href="#">MqlTradeRequest</a>
TRADE_ACTION_PENDING	Установить торговый ордер на совершение сделки при указанных условиях (отложенный ордер)	<a href="#">MqlTradeRequest</a>
TRADE_ACTION REMOVE	Удалить ранее выставленный отложенный торговый ордер	<a href="#">MqlTradeRequest</a>
TRADE_ACTION_SLTP	Изменить значения Stop Loss и Take Profit у открытой	<a href="#">MqlTradeRequest</a>

	позиции	
TRADE_RETCODE_CANCEL	Запрос отменен трейдером	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_CLIENT_DISABLES_AT	Автотрейдинг запрещен клиентским терминалом	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_CONNECTION	Нет соединения с торговым сервером	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_DONE	Заявка выполнена	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_DONE_PARTIAL	Заявка выполнена частично	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_ERROR	Ошибка обработки запроса	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_FROZEN	Ордер или позиция заморожены	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_INVALID	Неправильный запрос	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_INVALID_EXPIRATION	Неверная дата истечения ордера в запросе	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_INVALID_FILL	Указан неподдерживаемый <a href="#">тип исполнения ордера</a> по остатку	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_INVALID_ORDER	Неверный или запрещённый <a href="#">тип ордера</a>	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_INVALID_PRICE	Неправильная цена в запросе	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_INVALID_STOPS	Неправильные стопы в запросе	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_INVALID_VOLUME	Неправильный объем в запросе	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_LIMIT_ORDERS	Достигнут лимит на количество отложенных ордеров	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_LIMIT_VOLUME	Достигнут лимит на объем ордеров и позиций для данного символа	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_LOCKED	Запрос заблокирован для обработки	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_MARKET_CLOSED	Рынок закрыт	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_NO_CHANGES	В запросе нет изменений	<a href="#">MqlTradeResult</a>

TRADE_RETCODE_NO_MONEY	Нет достаточных денежных средств для выполнения запроса	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_ONLY_REAL	Операция разрешена только для реальных счетов	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_ORDER_CHANGED	Состояние ордера изменилось	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_PLACED	Ордер размещен	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_POSITION_CLOSED	Позиция с указанным <a href="#">POSITION_IDENTIFIER</a> уже закрыта	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_PRICE_CHANGED	Цены изменились	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_PRICE_OFF	Отсутствуют котировки для обработки запроса	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_REJECT	Запрос отвергнут	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_REQQUOTE	Реквота	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_SERVER_DISABLED_AT	Автотрейдинг запрещен сервером	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_TIMEOUT	Запрос отменен по истечению времени	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_TOO_MANY_REQUESTS	Слишком частые запросы	<a href="#">MqlTradeResult</a>
TRADE_RETCODE_TRADE_DISABLED	Торговля запрещена	<a href="#">MqlTradeResult</a>
TRADE_TRANSACTION_DEAL_ADD	Добавление сделки в историю. Осуществляется в результате исполнения ордера или проведения операций с балансом счета.	<a href="#">MqlTradeTransaction</a>
TRADE_TRANSACTION DEAL_DELETE	Удаление сделки из истории. Возможны ситуации, когда ранее исполненная сделка удаляется на сервере. Например, сделка была удалена во внешней торговой системе (бирже), куда она была выведена брокером.	<a href="#">MqlTradeTransaction</a>
TRADE_TRANSACTION DEAL_UPDATE	Изменение сделки в истории. Возможны ситуации, когда ранее исполненная сделка	<a href="#">MqlTradeTransaction</a>

	изменяется на сервере. Например, сделка была изменена во внешней торговой системе (бирже), куда она была выведена брокером.	
TRADE_TRANSACTION_HISTOR_Y_ADD	Добавление ордера в историю в результате исполнения или отмены.	<a href="#">MqlTradeTransaction</a>
TRADE_TRANSACTION_HISTOR_Y_DELETE	Удаление ордера из истории ордеров. Данный тип предусмотрен для расширения функциональности на стороне торгового сервера.	<a href="#">MqlTradeTransaction</a>
TRADE_TRANSACTION_HISTOR_Y_UPDATE	Изменение ордера, находящегося в истории ордеров. Данный тип предусмотрен для расширения функциональности на стороне торгового сервера.	<a href="#">MqlTradeTransaction</a>
TRADE_TRANSACTION_ORDER_ADD	Добавление нового открытого ордера.	<a href="#">MqlTradeTransaction</a>
TRADE_TRANSACTION_ORDER_DELETE	Удаление ордера из списка открытых. Ордер может быть удален из открытых в результате выставления соответствующего запроса либо в результате исполнения (заливки) и переноса в историю.	<a href="#">MqlTradeTransaction</a>
TRADE_TRANSACTION_ORDER_UPDATE	Изменение открытого ордера. К данным изменениям относятся не только явные изменения со стороны клиентского терминала или торгового сервера, но также и изменение его состояния при выставлении (например, переход из состояния <a href="#">ORDER_STATE_STARTED</a> в <a href="#">ORDER_STATE_PLACED</a> или из <a href="#">ORDER_STATE_PLACED</a> в <a href="#">ORDER_STATE_PARTIAL</a> и т.д.).	<a href="#">MqlTradeTransaction</a>

TRADE_TRANSACTION_POSITION	<p>Изменение позиции, не связанное с исполнением сделки. Данный тип транзакции свидетельствует именно о том, что позиция была изменена на стороне торгового сервера. У позиции может быть изменен объем, цена открытия, а также уровни Stop Loss и Take Profit. Информация об изменениях передается в структуре <a href="#">MqlTradeTransaction</a> через обработчик <a href="#">OnTradeTransaction</a>. Изменение позиции (добавление, изменение или ликвидация) в результате совершения сделки не влечет за собой появление транзакции TRADE_TRANSACTION_POSITION.</p>	<a href="#">MqlTradeTransaction</a>
TRADE_TRANSACTION_REQUEST	<p>Уведомление о том, что торговый запрос обработан сервером, и результат его обработки получен. Для транзакций данного типа в структуре <a href="#">MqlTradeTransaction</a> необходимо анализировать только одно поле - type (тип транзакции). Для получения дополнительной информации необходимо анализировать второй и третий параметры функции <a href="#">OnTradeTransaction</a> (request и result).</p>	<a href="#">MqlTradeTransaction</a>
TUESDAY	Вторник	<a href="#">SymbolInfoInteger</a> , <a href="#">SymbolInfoSessionQuote</a> , <a href="#">SymbolInfoSessionTrade</a>
TYPE_BOOL	bool	<a href="#">MqlParam</a>
TYPE_CHAR	char	<a href="#">MqlParam</a>
TYPE_COLOR	color	<a href="#">MqlParam</a>
TYPE_DATETIME	datetime	<a href="#">MqlParam</a>

TYPE_DOUBLE	double	<a href="#">MqlParam</a>
TYPE_FLOAT	float	<a href="#">MqlParam</a>
TYPE_INT	int	<a href="#">MqlParam</a>
TYPE_LONG	long	<a href="#">MqlParam</a>
TYPE_SHORT	short	<a href="#">MqlParam</a>
TYPE_STRING	string	<a href="#">MqlParam</a>
TYPE_UCHAR	uchar	<a href="#">MqlParam</a>
TYPE_UINT	uint	<a href="#">MqlParam</a>
TYPE ULONG	ulong	<a href="#">MqlParam</a>
TYPE USHORT	ushort	<a href="#">MqlParam</a>
UCHAR_MAX	Максимальное значение, которое может быть представлено типом uchar	<a href="#">Константы числовых типов</a>
UINT_MAX	Максимальное значение, которое может быть представлено типом uint	<a href="#">Константы числовых типов</a>
ULONG_MAX	Максимальное значение, которое может быть представлено типом ulong	<a href="#">Константы числовых типов</a>
UPPER_BAND	Верхняя граница	<a href="#">Линии индикаторов</a>
UPPER_HISTOGRAM	Верхняя гистограмма	<a href="#">Линии индикаторов</a>
UPPER_LINE	Верхняя линия	<a href="#">Линии индикаторов</a>
USHORT_MAX	Максимальное значение, которое может быть представлено типом ushort	<a href="#">Константы числовых типов</a>
VOLUME_REAL	Торговый объем	<a href="#">Ценовые константы</a>
VOLUME_TICK	Тиковый объем	<a href="#">Ценовые константы</a>
WEDNESDAY	Среда	<a href="#">SymbolInfoInteger</a> , <a href="#">SymbolInfoSessionQuote</a> , <a href="#">SymbolInfoSessionTrade</a>
WHOLE_ARRAY	Означает количество элементов, оставшееся до конца массива, то есть, будет обработан весь массив	<a href="#">Прочие константы</a>
WRONG_VALUE	Константа может неявно <a href="#">приводиться</a> к типу любого <a href="#">перечисления</a> .	<a href="#">Прочие константы</a>